UNIVERSITY OF OSLO
Department of informatics

Quality definitions and defect classes used in experiments on software inspection

Master thesis
<60 credits>

Morten Hordnes Bakken

<3. May 2009>

Summary

Quality in software engineering is a relevant topic because quality contributes to develop good, usable IT-systems and to satisfy customer expectations. To achieve this, we have to understand the term quality, what it means. To understand it, we have to have a clear definition of what quality is, which is easier said than done. Even though if we were about to understand it, we also have to know how to measure it, to determine whether one instance of quality is better than another. This is the quality issue in a nutshell.

Defects (or errors as he called them) were specified by Fagan in his original paper on formal inspection [fagan_76] as "any condition that causes malfunction or that precludes the attainment of expected or previously specified results. Thus, deviations from specifications are clearly termed errors." He made no further attempt to classify defects by type or severity. As far as I know, no widely adopted standard has been developed for classification of defects, but research on the field of inspection in later years often try to classify defects to some degree. This classification often tries to deal with the severity of the defects, its consequences for the development project and with where a type of defect often tends to appear. A good definition of what a defect is, and a solid classification scheme for defects is vital for performing good research on software inspection.

The objective of the research in this thesis was to study how quality is defined and how defects are classified and measured in a set of papers on formal experiments on software inspection.

In my opinion quality should get much more attention when researching software inspection than it has gotten in my selection of papers as software inspection is essentially a tool to enhance quality of a given software engineering artifact.

How much effort that were put into making solid classification schemes for defects varied wildly and some papers did not use classification of defects at all.
In my opinion a unified scheme for defect classification would serve the field of software inspection well.

Acknowledgements

I would like to thank my supervisor, Dag Sjøberg, for his guidance, support, contributions and discussions. I would also like to thank my good friend and fellow student Mili Orucevic for his comments, support and cooperation on this thesis, his help has been invaluable. Thanks to the students and employees at Simula Research Laboratory for making a nice work environment during the time I have worked on my thesis. Last but not least, thanks to my family, friends and my fiancée Stine for support and keeping out with me during this period.

Stabekk, May 2009
Morten H Bakken

Contents

1 Introduction 11
1.1 Motivation 11
1.2 Pair Programming 11
1.3 Problem formulation 12
1.4 Structure 13
2 Software Quality 15
2.1 The concept of quality 15
2.1.1 Popular view 16
2.1.2 Professional view 16
2.2 Measuring quality 17
2.3 Standards and measurements of Software Quality..... 18
2.3.1 ISO 9126 standard 18
2.3.2 Comparison of quality models 21
2.3.3 Other standards and models 22
2.4 Defects in software engineering artifacts 22
3 Related work 23
4 Research method 25
4.1 Systematic review 25
4.2 Review Protocol 26
4.3 How articles were selected and analyzed 26
4.3.1 Inclusion and exclusion criteria 26
4.3.2 Data sources and search strategy 27
4.3.3 Study identification and selection 27
4.3.4 Data extraction strategy 27
5 Analysis of the articles 29
5.1 Summary of the studies 29
5.1.1 Population selection 31
5.2 Quality Metrics 34
5.2.1 Analysis of findings regarding quality metrics 34
5.3 Defect classification 35
5.4 Distribution of defect classifications used 36
5.4.1 Analysis of defect classifications used 36
5.4.2 Defect classification by severity 36
5.4.3 Defect classification by type 37
6 Discussion 39
6.1 Discussion regarding findings related to research question 1 39
6.2 Discussion regarding findings related to research question 2 40
6.2.1 Classification of defects by severity 40
6.2.2 Classification of defects by type 41
7 Threats to validity 43
7.1 Selection of Articles 43
7.2 Data extraction 43
8 Conclusion 45
8.1 Objective of research 45
8.2 Findings 45
8.3 Future work 46

Appendix A – Analysis of the selected articles ……47
The empirical investigation of perspective based reading ……...……...47
A controlled experiment to assess the effectiveness of inspection meetings ……...…47
Using inspection data for defect estimation ……..……...48
Evaluating the accuracy of defect estimation models based on insepction data from two inspection cycles ……..48
Investigating the influence of inspector capability factors with four inspection techniques on inspection performance ……49
Investigating the cost-effectiveness of reinspections in software development ..……49
Further investigations into the development and evaluation of reading techniques for object-oriented code inspection ……..……50
Object-oriented inspection in the face of delocalization ……..…51
A replicated experiment to assess requirements inspection techniques ……..……51
Assessing software review meetings: A controlled experiment using CSRS …….……52
A comparison of tool-based and paper-based software inspection ……..……...…52
Further experiences with scenarios and checklists ……..…..…53
Comparing detection methods for software requirements inspections: a replicated experiment ……..……53
Assessing software review meetings: Results of a comparative analysis of two experimental studies ……..54
Perspective-based usability inspection: An empirical validation of efficacy ..……54
Appendix B – Disclaimer on collaborative work in master thesis …….55
Literature list ……..……57

List of tables

Table 1: Comparison between criteria/goals of the McCall, Boehm and ISO9126 quality models ……..21
Table 2: Summary of all the studies ……29
Table 3: Quality metric per article ……...34
Table 4: Defect classification per article ……...……...35

List of figures

Figure 1: Main factors in ISO 9126 ……19
Figure 2: Main and sub-factors in ISO9126 ……19
Figure 3: Number of population types in the experiments ……..31
Figure 4: Number of subjects (professionals) used in the experiments ……...…32
Figure 5: Number of subjects (students) used in the experiments ……...……33
Figure 6: Distribution of defect classifications used ……...36

1 Introduction

1.1 Motivation

Quality in software engineering is a relevant topic because quality contributes to develop good, usable IT-systems and to satisfy customer expectations. To achieve this, we have to understand the term quality, what it means. To understand it, we have to have a clear definition of what quality is, which is easier said than done. Even though if we were about to understand it, we also have to know how to measure it, to determine whether one instance of quality is better than another. This is the quality issue in a nutshell. Quality is such a wide defined term in computer science, especially in software engineering. Whenever someone mentions quality, they tend to use their own instances and definitions of quality. Still, we don't have a clear definition of what quality is. When two or several experiments in the same research area use different quality definitions, it is nearly impossible to compare these measurements and results they provide to each other.
Measuring length, height, age etc. is easy, because they have only one measurement to consider, and people have a common understanding of how to measure these terms. Length and height are measured in meters, age in years, but when it comes to software quality measurement; there are no common understandings of how to measure quality. Some standards have appeared like ISO9126, just to mention one, but it is not easy to change a whole industry over night. The main goal of this Master's thesis is to find out how articles describing the effect of pair programming have defined and measured software quality. Are there some equality in these definitions and measurements, or have the articles focused on different measurements of quality?

1.2 Software inspection

Software engineering artifacts (both documentation and source code) have been read and searched for defects as long as they have been produced. In 1976 however, Fagan, proposed a structured way to do this [fagan_76]. This is generally referred to as a "Formal Technical Review" or FTR [porter_et_al_97]. The process consists of five steps:

- Overview. The author presents an overview of the scope and purpose of the work product.
- Preparation. Reviewers analyze the work product with the goal of understanding it thoroughly.
- Inspection meeting. The inspection team assembles and the reader paraphrases the work product. Reviewers raise issues that are subsequently recorded by the scribe.
- Rework. The author revises the work product.
- Followup. The moderator verifies the quality of rework and decides if reinspection is required.

This process has been the base of more or less all inspections carried out since Fagan published his paper.

Even though the process still is quite equal to what Fagan proposed, several refinements have been proposed through the years. Some researchers say that an ad-hoc approach is the best, where you just read through the material and search for defects, others say that a detailed checklist is better and then some say that scenarios are the best. Different reading techniques have also been proposed. When inspections first started, programming languages where procedural, meaning that reading sequentially was effective. For object-oriented code this is harder so some researchers have suggested that you follow the flow of execution when reading such code [dunsmore_et_al2_00] [dunsmore_et_al_02]. For requirements documentation some have proposed a perspective based reading technique, where you assume a perspective, for instance as a tester, a developer and so on.

Some researchers have also proposed to skip the meeting step in Fagan's original process, as their experience is you "lose" more defects during these meetings than you "gain" (meaning that defects proposed by individual inspectors are turned down during the meeting) [Johnson_et_al_97]. Meetings however are good for filtering out false positives, training novices and increasing confidence of the team, so some researchers say that they might serve a purpose afterall [bianchi_et_al_01.pdf].

1.3 Problem formulation

This thesis studies the concept of the term quality used in a set of selected research papers on software inspection. To best answer this it is also necessary to find out how these papers define and classify a defect, since detecting them is the purpose of software inspection. Reducing the amount of defects in a software engineering artifact is a key activity for improving the quality of it. I present an overview that characterizes what authors and researchers call quality and how they define defects in these research papers. On the basis of this overview, other authors and researchers may decide further research for improving and narrow the use of the terms quality and defect.

The objective of the investigation is to get an overview of how quality and defects is defined, described and measured in research papers on software inspection.

In order to address these issues, I have analyzed a set of research papers on software inspection. The set consist of 15 articles, which are selected from a larger set (214 articles). In section 4.2 review protocol, it is explained how these 15 articles were selected. The data collected during analysis of these articles, was used to answer the following research questions:

RQ1: *How is quality defined in a set of research papers on software inspection?*
RQ2: *How are defects defined and classified in a set of research papers on software inspection?*

1.4 Structure

The first Section gives a brief introduction to the thesis, as well as a short introduction to pair programming and the problem formulation. Section 2 “Software Quality” investigates the history of quality, what quality is, how it is understood up to today’s date, and describes different standards of quality. Section 3 “Related Work” sums up relevant and related work to this thesis done by different authors. Section 4 “Research Methods” describes the systematic review, and review protocol which describes how the articles were selected, analyzed and processed, as well as describing the data extraction strategy. Section 5 “Analysis of the articles” presents the analysis of the articles, raw and processed data found. Detailed analysis regarding quality metrics used and classification of quality metrics. Section 6 “Discussion” discusses the findings done in the previous Section, what authors call quality, their affiliations and more. Section 7 “Threats to Validity” addresses the issue regarding validity in this thesis, and in Section 8 “Conclusion” I present the conclusion of this thesis where I discuss the findings, what is learned from the thesis and how it can be used in the future by others.

2 Software Quality

2.1 The concept of quality

One historian of ideas suggests that it was Plato who should be credited with inventing the term quality.

The more common a word is and the simpler its meaning, the bolder very likely is the original thought which it contains and the more intense the intellectual or poetic effort which went into its making. Thus, the word *quality* is used by most educated people every day of their lives, yet in order that we should have this simple word Plato had to make the tremendous effort (it is perhaps the greatest effort known to man) of turning a vague feeling into a clear thought. He invented a new word 'poiotes', 'what-ness', as we might say, or 'of-what-kind-ness', and Cicero translated it by the Latin 'qualitas', from 'qualis'. [bar_88]

Plato also debated over the definition of quality through his dialogs. One example is the dialog in the Greater Hippias, where there is a dialog between Socrates and Hippias. Socrates, after criticizing parts of an exhibition speech by Hippias as not being fine, asks the question "what the fine is itself?".
Even though the word "quality" has not existed as long as humans have, it has always been around.
One of the earliest quality movements can be traced back to Roman crafters, such as blacksmiths, shoemakers, potters etc. They formed groups which they called *collegiums*, emphasizing in its etymology a group of persons bound together by common rules or laws. This helped the crafters/members to achieve a better product, because of the tighter collaboration with each other. Achieving a better product can be interpreted as gaining better product quality, within a marked [eps_91]. This was also a phenomenon in the 13th century, where European groups/unions, called guilds, were established with the same purpose as the Romans collegiums. These groups had strict rules for product and service quality, even though they didn't call it that. Craftsmen themselves often placed a mark on the goods they produced, so that it could be tracked back to crafter in case it was defect, but over time this came to represent craftsman's good reputation. These marks (inspection marks, and master-crafter marks) served as proof of quality (like today's ISO) for customers throughout medieval Europe, and were dominant until the 19th century. This was later transformed into the factory system, which emphasized product inspection. In the early 20th century the focus on processes started with Walter Shewhart, who made quality relevant not only for the finished product but for the processes as well [asq_1]. Shortly after the WW2 in 1950, W. Edwards Deming provided a 30 day seminar in Japan for Japanese top management on how to improve design, product quality, testing and sales. Beside Deming, Dr. Joseph M. Juran also contributed to raise the level of quality from the factory to total organization. Eventually the U.S.A. adopted this method, from the Japanese, and expanded it from emphasizing only statistics, to embrace the entire organization. This became known as Total Quality Management (TQM). In the late 1980s, the International Organization for Standardization (ISO) published a set of international standards for quality management and quality assurance,

called ISO 9000. This standard underwent a major revision in 2000, now this includes ISO 9000:2000 (definitions), ISO 9001:2000 (requirements) and ISO 9004:2000 (continuous improvement). [asq_2]

Even today, decades after the quality standards entered the market; the term quality is an ambiguous term. Many organizations, researchers and authors have trough out the years defined the term quality as they envision it. Phil Crosby [cro_79] defines it as "conformance to user requirements", Watts Humphrey [hum89] refers to it as "achieving excellent levels of fitness for use", IBM use the phrase "market-driven quality", and the Baldrige criteria is similar to IBMs "customer-driven quality". The most recent definition of quality is found in ISO9001-00 as "the degree to which a set of inherent characteristics fulfills requirements". Kan has also a definition on quality, which is perhaps broader than those defined in the past; *"First, quality is not a single idea, but rather a multidimensional concept. The dimensions of quality include the entity of interest, the viewpoint on that entity, and the quality attributes of that entity. Second, for any concept there are levels of abstraction; when people talk about quality, one party could be referring to it in its broadest sense, whereas another might be referring to its specific meaning. Third, the term quality is a part of our daily language and the popular and professional uses of it may be very different"[kan_04]*.

Kan [kan_04] describes quality from two viewpoints, the popular and the professional.

2.1.1 Popular view

As one may see, the notation of "quality" is not as simple as it may seem. In everyday life people use the term quality, as an intangible trait; it can be discussed, felt and judged, but cannot be weighed or measured. Too many people use the terms *good quality, bad quality* without even wanting to define what they mean by good/bad quality, I am not even sure that they themselves know what they mean by it, but their view on quality is simply; "I know it when I see it". Another popular view is that quality often is associated with luxury and class. Expensive and more complex products are regarded as products with higher quality. If we take cars as an example, most of us will agree that BMW is somehow a higher quality car than Honda, but according to Initial Quality Study (IQS), [nett_0607] Ranking made in 2007 where they surveyed new car owners about their cars, to find out problems per 100 vehicles, Honda scored 108/100 and BMW scored 133/100. 25 more defects were found per 100 vehicles in BMW then in Honda. Since BMW is a more expensive car, and holds a higher status in the western world, it is considered to be a car with more quality. Simple, inexpensive products can hardly be classified as quality products.

2.1.2 Professional view

The opposite of the popular view is the professional view. Because of the vagueness, misunderstanding and misconception of the popular view, the quality improvement in the industry are not evolving at a great pace. Due to this quality must be described in a workable definition. As mentioned earlier, Crosby [cro_79] defines quality as "conformance to

requirements", and Juran and Gryna [jur_gry_70] defines quality as "fitness to use". These definitions are essentially similar and consistent, and are there for adopted by many professionals.

"Conformance to requirements" states that requirements must be clearly stated in a way that they cannot be misunderstood. Everything that is non-conformant is regarded as defects or lack of quality; because of this, measurements are regularly taken under the production and development processes to determine the level of conformance. For example a new light bulb enters the market, and one of the requirements is that it should last at least 300 hours. If it fails to do so, will this be seen as a lack of quality, it does not meet the quality requirements that are set and there for should be rejected. Taking this in regard, if a Honda conforms to all requirements that are set for it, then it is still a quality car. The same thing also counts for BMW, if all requirements are fulfilled, then it is a quality car. Even though these two cars are different in many ways, comfort, economics, style, status, etc. If both measure up to the standard set for them, then both are quality cars.

The term "fitness for use" implies a more significant role for the customer's requirements and expectations, then the "conformance to requirements" definition. Different customers have different views and different use of the product. This means that the product must have multiple elements of fitness for use. Each of these elements is a quality characteristic, and can be categorized as parameters for fitness for use. The two most important parameters are "quality of design" and "quality of conformance". Quality of design can be regarded as determination of requirements and specification. In popular terminology these are known as grades or models, and are often linked to purchasing power. Taking the cars example again, all cars are designed to transport one or several persons from A to B, but models differ in several things. These things can be size, comfort, style, economics, status, performance etc. Quality of conformance is simply the conformance to the requirements set by the quality of the design.

2.2 Measuring quality

To measure something, one must know what "it" is, then develop a metric that measures "it". This applies to the quality issue as well. If there was a simple way to measure quality, it would already been known and used, but since there are many definitions on what quality is, the measurements varies a lot. Even though several standards exist today, the industry has not been able to adopt it, not to mention the academic field as well. At this point most companies within the computer sector have some form for quality assurance. They define quality according to what they believe it is, and measure it the same way. Not many have adopted the international standards yet. Since this thesis is not about what quality measurements are used in the industry, I will not be digging more into this subject but rather investigate how this is done in an academic setting, regarding experiments about pair programming efficiency.

The bottom line is that since few companies follow same standards for what quality is and how to measure it, it is extremely difficult to compare both processes and products with different definitions and i.e. claim that one product/process is better than another.

2.3 Standards and measurements of Software Quality

2.3.1 ISO 9126 standard

ISO 9126 is an international standard for the evaluation of software. One of the fundamental objectives of this standard is to address human biases that can affect the delivery and perception of a software project. The standard is divided into four parts which addresses, respectively, the following subjects;

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics

Quality model

The ISO 9126-1 software quality model identifies 6 main quality characteristics (figure 1), namely:

- **Functionality**
  A set of attributes that relate to the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs [sqmap].

- **Reliability**
  A set of attributes that relate to the capability of software to maintain its level of performance understated conditions for a stated period of time [sqmap].

- **Usability**
  A set of attributes that relate to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users [sqmap].

- **Efficiency**
  A set of attributes that relate to the relationship between the level of performance of the software and the amount of resources used, under stated conditions [sqmap].

- **Maintainability**
  A set of attributes that relate to the effort needed to make specified modifications [sqmap].

- **Portability**
  A set of attributes that relate to the ability of software to be transferred from one environment to another [sqmap].

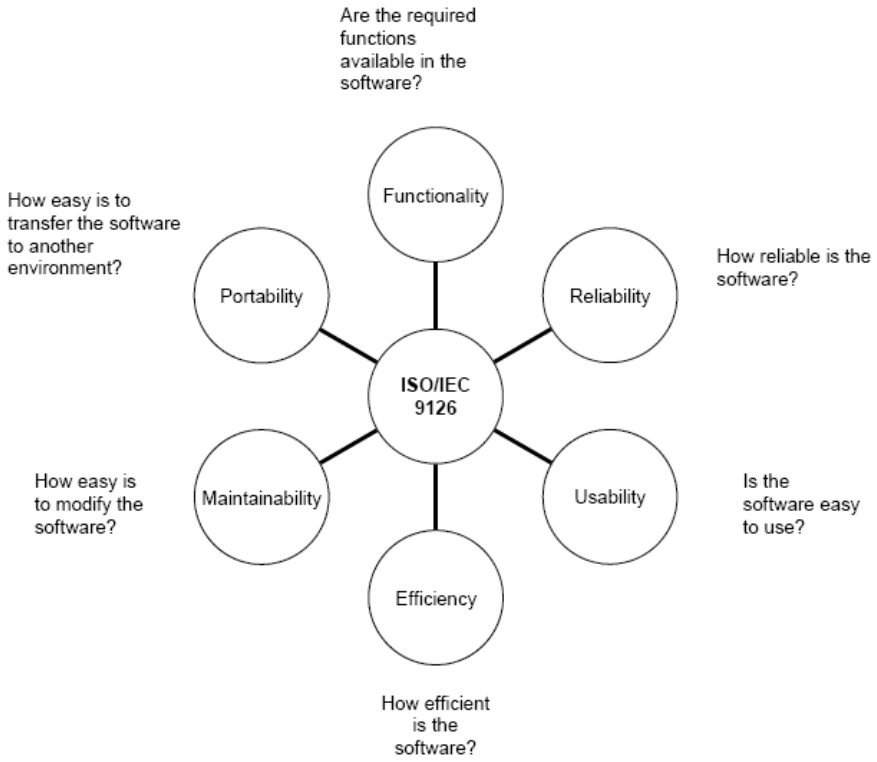

Figure 1: Main factors in ISO 9126

Each factor in the ISO 9126 contains several sub-factors, which are all shown in figure 2.

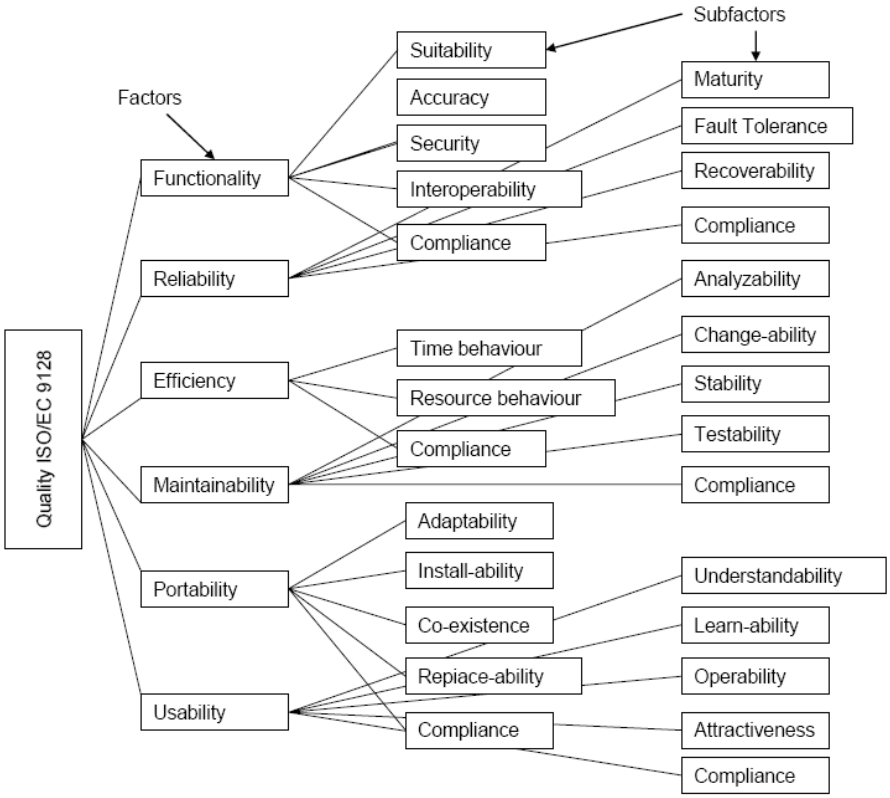

Figure 2: Main and sub-factors in ISO9126

If one is interested in a more detailed explanation on all these factors and sub-factors, one can visit the www.iso.org site.

External metrics

External metrics are applicable to running software.

Internal metrics

Internal metrics are those which do not rely on software execution (static measurements)

Quality in use metrics

Quality in use metrics, are only available when the final product is used in real conditions.

Ideally, the internal quality determines the external quality and this one determines the results of quality in use [scalet_etal_00].

2.3.2 Comparison of quality models

The ISO 9126 standard is based on McCall and Boehm models. Besides being structured in basically the same manner as these models (figure 2), ISO 9126 also includes functionality as a parameter, as well as identifying both internal and external quality characteristics of software products. Table 1 presents an comparison between the quality models.

Table 1 – Comparison between criteria/goals of the McCall, Boehm and ISO9126 quality models [sqmap]

Criteria/goals	McCall, 1977	Boehm, 1978	ISO 9126, 1993
Correctness	*	*	Maintainability
Reliability	*	*	*
Integrity	*	*	
Usability	*	*	*
Efficiency	*	*	*
Maintainability	*	*	*
Testability	*		Maintainability
Interoperability	*		
Flexibility	*	*	
Reusability	*	*	
Portability	*	*	*
Clarity		*	
Modifiability		*	Maintainability
Documentation		*	
Resilience		*	
Understandability		*	
Validity		*	Maintainability
Functionality			*
Generality		*	
Economy		*	

2.3.3 Other standards and models

Both old and new standard and models are listed below. I will not elaborate more on these models, as it is not a part of my thesis. One can read more about these models elsewhere. The reason I mention them is to point out what standards and models are out there which help improve quality definitions and measurements. Also worth noticing is that some of these models build upon one other. As listed in [sqmap]:

- McCall's Quality Model
- Boehm's Quality Model
- FURPS / FURPS +
- Dromey's Quality Model
- ISO – standards
  - ISO 9000
  - ISO 9126
  - ISO / IEC 15504 (SPICE)
- IEEE
- Capability Maturity Model(s)
- Six Sigma

...

2.4 Defects in software engineering artifacts

Defects (or errors as he called them) were specified by Fagan in his original paper on formal inspection [fagan_76] as "any condition that causes malfunction or that precludes the attainment of expected or previously specified results. Thus, deviations from specifications are clearly termed errors." He made no further attempt to classify defects by type or severity. As far as I know, no widely adopted standard has been developed for classification of defects, but research on the field of inspection in later years often try to classify defects to some degree. This classification often tries to deal with the severity of the defects, its consequences for the development project and with where a type of defect often tends to appear. A good definition of what a defect is, and a solid classification scheme for defects is vital for performing good research on software inspection.

3 Related work

I have not been able to find any research resembling what I have undertaken with this master thesis. Exploring the quality and defect classification used in controlled experiments on software inspection is groundbreaking work.

Sjøberg et al [sjøberg_et_al_05] did a survey of controlled experiments in software engineering in 2005, but they did not focus only on software inspection, neither did they focus on quality metrics or defect classification.

Much research has been done in the field of software inspection however. Formal software inspection was incepted in 1976 by Fagan, so it is an ancient field in the world of software engineering. In the search done by Sjøberg et al [sjøberg_et_al_05] they found 103 formal experiments. 37 of those were on some form of inspection.

I have not done any exhaustive searches but a simple search for “inspection” gives 6641 hits in the ACM portal and a search for “software inspection” gives 4832 hits.

Much of the later research on the field seems to be in the direction of the effectiveness of new variants of Fagan’s original method, but the measurements that is done is perhaps a bit crude, and does not give a complete picture of the capability of the different methods. I will get back to this in later chapters.

4 Research method

As the purpose of this research is to examine how quality is defined and measured in a set of software engineering experiments, a systematic review was chosen as the research method. Before examining the selection of articles, I carried out several literature investigations about the term quality, how it is defined, how it is measured, what kind of standards exist today etc.

4.1 Systematic review

Systematic review is the means of evaluating and interpreting all available research relevant to a particular research question, topic area or phenomenon of interest [kitchenham_04]. Furthermore she also states that the aim of systematic review is to present a fair evaluation of a research topic using a trustworthy, rigorous, and auditable methodology. A systematic review must be undertaken in accordance with a predefined search strategy [kitchenham_04].

The major advantage of systematic review is that they provide information about the effects of some phenomenon across a wide range of settings and empirical methods [kitchenham_04].

Important features of a systematic review [kitchenham_04]:

- Systematic reviews start by defining a review protocol that specifies the research questing being addressed and the methods that will be used to perform the review
- Systematic reviews are based on a defined search strategy that aims to detect as much of the relevant literature as possible.
- Systematic reviews document their search strategy so that readers can access its rigor and completeness.
- Systematic reviews require explicit inclusion and exclusion criteria to assess each potential primary study.
- Systematic reviews specify the information to be obtained from each primary study including quality criteria by which to evaluate each primary study
- A systematic review is a prerequisite for quantitative meta-analysis.

4.2 Review Protocol

General/Background

The reasons why I have decided to include a review protocol are to avoid research bias and to avoid the analysis to be driven by researcher's (my) expectations [SEG_97]. Also, with a strict structure of how to analyze the articles, it will be easier to compare the articles to each other.

Review supervisor

Name	Current position	Skills relevant to SLR	Role
Dag Sjøberg	Research Director at Simula Research Laboratory	Research methods for empirical software engineering, theoretical foundations for empirical software engineering	Master thesis supervisor and mentor

Research questions

RQ1: *How is quality defined in a set of research papers on software inspection?*
RQ2: *How are defects defined and classified in a set of research papers on software inspection?*

4.3 How articles were selected and analyzed

The selection of the articles is done in a prior study, [sjøberg_et_al_05], which was a survey of controlled experiments in software engineering. Since the selection has been done without my involvement I will just replicate the method used in [sjøberg_et_al_05].

4.3.1 Inclusion and exclusion criteria

The authors of [sjøberg_et_al_05] selected 103 controlled experiments from 5453 scientific articles published in a selection of journals between 1993 and 2002. The inclusion criteria were that the article was a controlled experiment. Excluded from the search were editorials, prefaces, article summaries, interviews, news, reviews, correspondence, discussions, comments, reader's letters, and summaries of tutorials workshops, panels, and poster sessions.

4.3.2 Data sources and search strategy

The authors selected all scientific articles published in EASE, EMSE, ICSE, IEEE Computer, IEEE Software, ISESE, IST, JSME, JSS, METRICS, SP&E, TOSEM and TSE between 1993 and 2002.

4.3.3 Study identification and selection

All of the scientific articles (5453) had their title and abstract systematically read by one researcher. If it was unclear from the title or abstract whether a controlled experiment was described, the entire article was read by both the same researcher and another person in the project team. In the end, 103 articles were selected. The process of identifying relevant articles was not as straightforward as it might seem, as several authors claimed to report on an experiment even though no treatment was applied in the study.

Of these 103 experiments, 15 were randomly selected for me with the only criteria being that it was on software inspection.

4.3.4 Data extraction strategy

I will extract all the data myself, and have not the possibility to send these extractions to another person for verification, which would be the optimal thing to do. Ideally one external person should have reviewed the data extractions, to verify and to exclude research bias. I will probably use this method when I write future papers, if the settings allow it.

In this assignment I will focus on extracting the following data:

- *General information*
  - Title, authors, experiment setting
- *Main Research Questions*
  - Quality descriptions
  - Defect definitions and classifications
- *Other Specific information*

The main goal of this paper is to find out how quality and defects are defined and also if the definitions are similar to each other.

5 Analysis of the articles

Every article is analyzed according to the review protocol, with the purpose of being able to compare the results to each other. For those who are interested in reading a complete and detailed analysis of the articles, this can be found in Appendix A at the end of the thesis.

This section summarizes the findings done by the analysis. At first I'll presents an overview of all the articles in general. This to get the reader familiarized with all of the articles analyzed. The article's name can be found in the literature list, following the reference. Then I will present all quality and defect classification findings per article. Afterward I will provide information about any common use of quality attributes and defect classifications.

5.1 Summary of the studies

Table 2: Summary of all the studies

Study	Subjects	Total amount of subjects	Study setting
Basili_96_b	Professionals	26	NASA employees using a perspective based reading technique vs. their usual technique
Bianchi_et_al_01	Students	100+	Groups of students doing individual and then group inspection
Biffl_00	Students	169	Students doing inspection of an object representative of real world development specifications
Biffl_01	Students	169	Students doing inspection of an object representative of real world development specifications
Biffl_et_al	Students	177	Students doing checklist and scenario-based inspection
Biffl2_01	Students	169	Students doing two cycles of inspection, to measure the effectiveness of reinspection
Dunsmore_et_al_02	Students	69	Students performing inspection with checklist, use-cases or a systematic technique. Compare effectiveness
Dunsmore_et_al2_00	Students	47	Students performing inspection with three techniques, ad-hoc, checklist and scenario-based
Fusaro_et_al_97	Students	30	Students performing inspection with three techniques, ad-hoc, checklist and scenario-based
Johnson_et_al_97	Students	72	Students doing real group review and nominal group review assisted by a tool called CSRS.
Macdonald_et_al_98	Students	43	Students split into two groups, one doing tool-based inspection

			and the other paper-based inspection
Miller_et_al_98	Students	50	Students doing inspection with checklist and scenario based inspection
Porter_et_al_95	Students	48	16 groups of students doing two inspections with a combination from three possible techniques, ad-hoc, checklist and scenario-based
Porter_et_al_97_b	N/A	N/A	Comparative analysis of two experiments on inspection
Zhang_et_al_99	Students/professionals	7/24	Pilot experiment with students, then real experiment with professionals. Usability inspection with different perspectives.

5.1.1 Population selection

As we can see in fig. 3, 13 of the experiments uses students as subjects and 2 of the experiments used professionals. This is quite common as an experiment can be implemented as part of university curriculum or students might be willing to participate in an experiment for no or very little compensation. Professionals however will often require monetary compensation for participating in an experiment and it is harder to coordinate because they have a job to attend to.

Professionals will of course give the best picture of how professionals respond to treatments in an experiment, but students are the professionals of tomorrow so their responses should also have some validity.

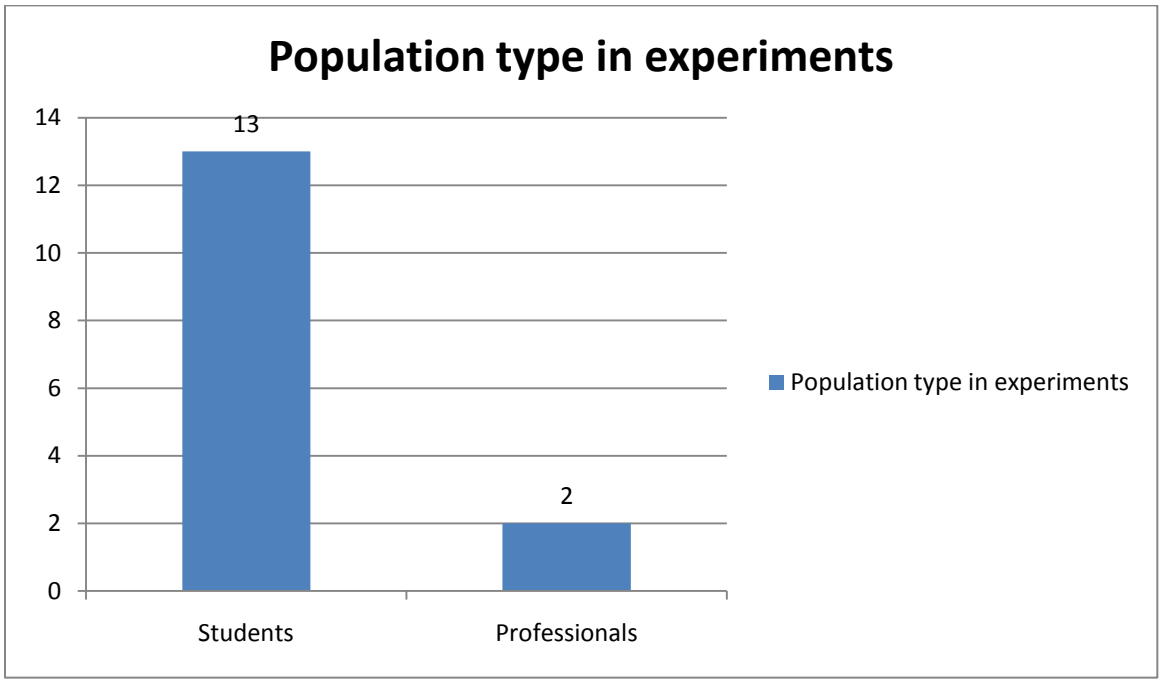

Figure 3: Number of population types in the experiments

Fig. 4 shows many subjects participated in the experiments where professionals were used. The numbers is quite similar, and they reflect the size of an average development department in an IT organization well.

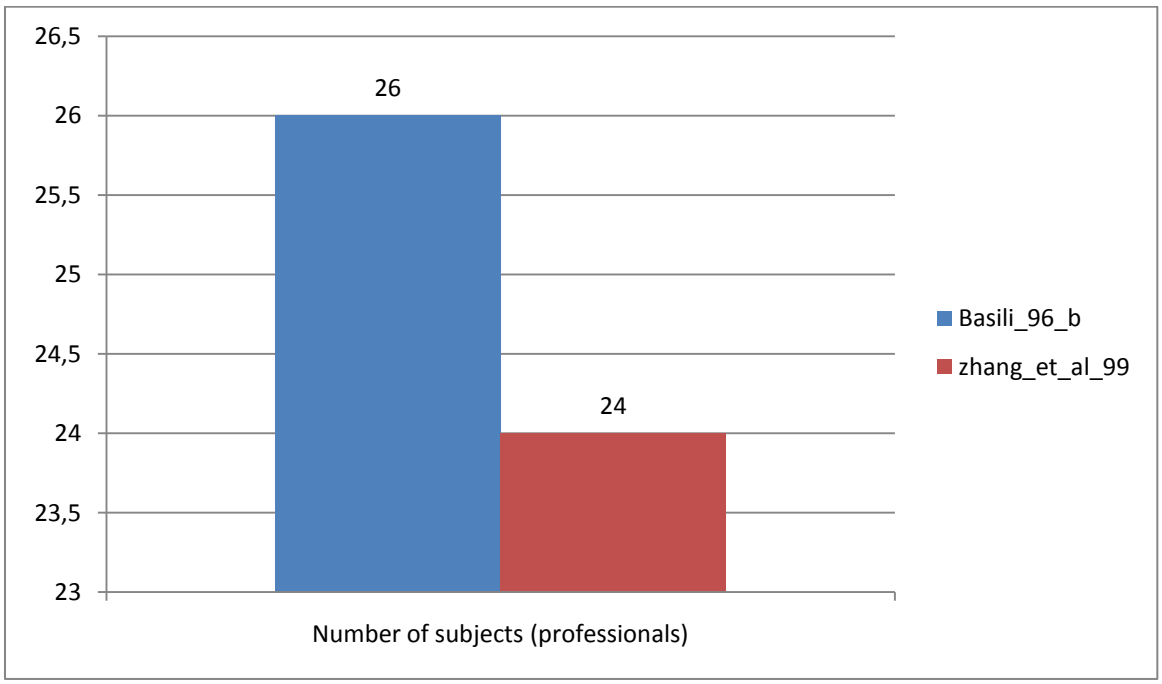

Figure 4: Number of subjects (professionals) used in the experiments

Fig. 5 shows the distribution of subjects in those experiments that used students. As expected it varied quite a lot. Some of the experiments used the participants of a large two-semester workshop on a university and integrated the experiment as a part of that. None of the experiments had so few subjects that the results should be completely irrelevant.

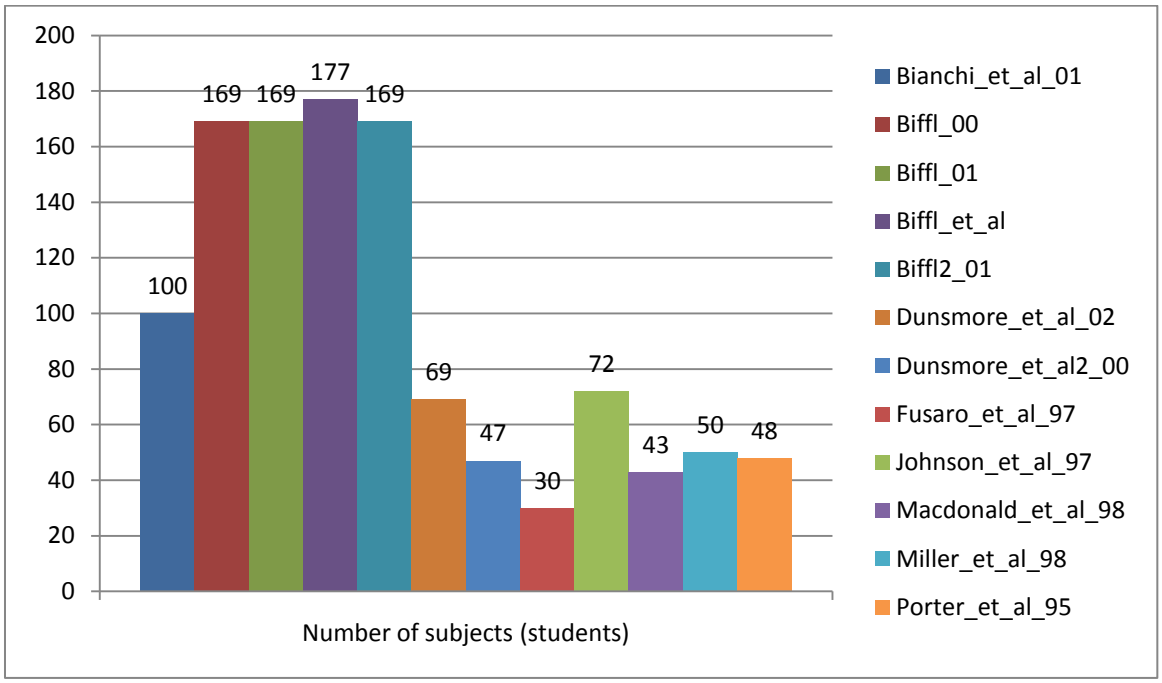

Figure 5: Number of subjects (students) used in the experiments

5.2 Quality Metrics

A metric is a measurement of some property of a piece of software, its specification or the process.
Quantitative methods have proved to be very powerful in other science, computer scientists have worked hard to bring software development similar approaches. In the set of articles provided for this thesis, just a few of the authors used any quality metrics. Every metric used is mentioned and explained in table 3. The table shows which article use which metric, and how the author define and measure this metric.

Quality metric usage per article

Table 3: Quality metric per article

Study	Quality Metric	Definition and measurement
Basili_96_b	None	None
Bianchi_et_al_01	None	None
Biffl_00	Project Cost Number and type of defects	Time to complete project Count of defects distributed by severity
Biffl_01	None	None
Biffl_et_al	None	None
Biffl2_01	Project Cost Number of defects	Time to complete project Count of defects
Dunsmore_et_al_02	None	None
Dunsmore_et_al2_00	None	None
Fusaro_et_al_97	None	None
Johnson_et_al_97	None	None
Macdonald_et_al_98	Efficiency	Brief mention that defects can affect efficiency, no complete definition or measurement
Miller_et_al_98	None	None
Porter_et_al_95	None	None
Porter_et_al_97_b	None	None
Zhang_et_al_99	Usability	A list of 9 attributes that relate to a systems usability. Subjective measurement, "experienced usability"

5.2.1 Analysis of findings regarding quality metrics

Out of the 15 experiments only in 4 of them is any quality attributes used. The attribute project cost is discussed twice, number of defects twice and efficiency and usability is used in one experiment each. The rest of the papers do not focus on how software inspection can affect the quality of software engineering artifacts at all. There is not much data regarding my first research question: "*How is quality defined in a set of research papers on software inspection?"* I will discuss these finding to greater extent in chapter 6.

5.3 Defect classification

Here I will give a list of how the defects are classified in the different experiments. Under the column defect classification I will report if defect classification is used and under the column definition I will report how it is defined if it is defined.

Table 4: Defect classification per article

Study	Defect classification	Definition
Basili_96_b	Defect classification by type and severity	Nothing defined or used in the experiment, but it is mentioned as a target for future work
Bianchi_et_al_01	None	None
Biffl_00	None	None
Biffl_01	Defects classified by type	Either missing, unnecessary, wrong, ambiguous or inconsistent information.
Biffl_et_al	Defects classified by severity	Some discussion that one of the techniques can find the most severe defects, but this is not measured
Biffl2_01	Defects classified by severity	Four severity levels
Dunsmore_et_al_02	Defects classified by type	Some mention that a defect can be of different types, but not much elaboration on this. They separate between localized and delocalized effects, caused by object-oriented code.
Dunsmore_et_al2_00	Defects classified by type	Defects are classified by how they appear in the code. Local or non-local. Whether they are in small or large methods, caused by wrong use of inheritance and so on.
Fusaro_et_al_97	Defects classified by type	Taxonomy by [Schneider_et_al_92]
Johnson_et_al_97	Defects classified by type and severity	It is mentioned in the paper that defects found is of certain types, but there are no exhaustive list of types given or any discussion about defect types. The software they use to inspect has an attribute for severity level of defect but this is not further discussed in the paper
Macdonald_et_al_98	Defects classified by type and severity	They make a difference between defects that make the program erroneous and defects that affect the quality of the program. It is also mentioned that there are different levels of defects related to how hard they are to find, but no list of these levels are given
Miller_et_al_98	Defects classified by type	Taxonomy by [Schneider_et_al_92]
Porter_et_al_95	Defects classified by type	Taxonomy by [Schneider_et_al_92]
Porter_et_al_97_b	Defects classified by type	Discussion on whether there exists a type of defects that are easier to find with group inspection
Zhang_et_al_99	Defects classified by severity	Usability problems as defects are called in this paper is ranked by Nielsen's rating scale [Nielsen_93] on a 5 level scale.

5.3.1 Distribution of defect classifications used

Of the 15 studies only 2 used no defect classification at all. The rest used defect classification to some extent. Fig.6 shows that 3 of the papers classified defects by severity only, 10 of the papers classified defects by type only and 3 of the papers classified defects by both severity and type.

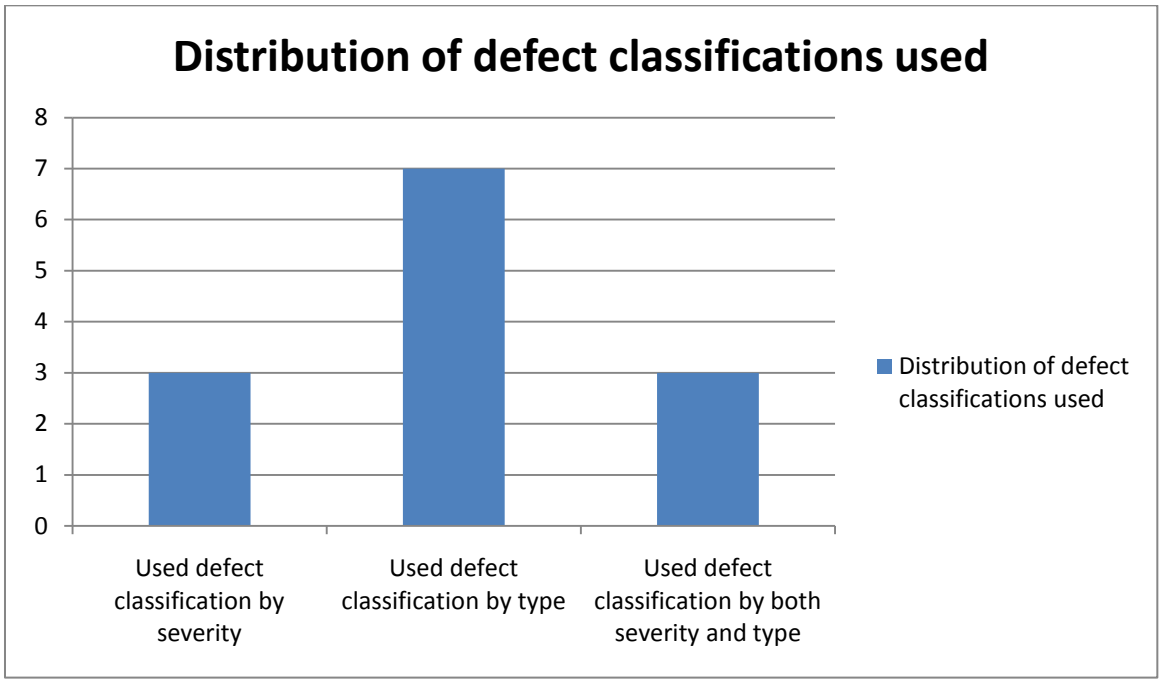

Figure 6: Distribution of defect classifications used

5.3.2 Analysis of defect classifications used

In this section I will give an overview of how the different experiments classify defects, whether it is by severity or type.

5.3.2.1 Defect classification by severity

The authors of the experiments that classified defects by severity either used a small scale with 3 to 5 levels ranging from minor irrelevant defects to critical defects, or they just mentioned that there exists defects of different severity without making a framework for describing the severity.

5.3.2.2 Defect classification by type

Those authors who classified defects by type used varying types. The types used did of course differ depending on what kind of software engineering artifact that was inspected. For requirements documentation, there was three papers that used a common set of defect types, a taxonomy proposed by Schneider et. al in their paper "An experimental study of fault detection in user requirements documents" [Schneider_et_al_92]. The taxonomy looks like this:

Class 1 Faults. Missing Information
(a) Missing Functionality or Missing Feature (MF). Information describing the desired internal operational behavior of the system has been omittedfrom the URD.
(b) Missing Interface (Ml). Information describing how the proposed system will interface and communicate with objects outside the scope of the system has been omitted from the URD.
(c) Missing Performance (MP). Information describing the desired performance specifications has either been omitted or described in a way that is unacceptable for acceptance testing.
(d) Missing Environment (ME). Information describing the required hardware /software/database/personnel environment in which the proposed system will run has been omitted from the URD.

Class 2 Faults. Wrong Information
(e) Ambiguous Information (WA). An important term, phrase, or sentence essential to an understanding of system behavior has either been left undefined or defined in a way that can cause confusion and misunderstanding. (Note, these are not merely language ambiguities such as an uncertain pronoun reference, but ambiguities about the actual system and its behavior.)
(f) Inconsistent Information (Wl). Two sentences contained in the URD directly contradict each other or express actions that cannot both be correct or cannot both be carried out.

This is a rather good taxonomy that should cover a lot of defects related to requirements documentation. The other papers that experimented on inspection of documentation and classified defects by type, used similar types to the ones found in Schneider's taxonomy.

Another type of software engineering artifact that was experimented on was source code. The experiments on these artifacts that defined defect classes by type all used their own types and none of them had a very well defined and exhaustive taxonomy of defect types. [Dunsmore_et_al_02] and [Dunsmore_et_al2_00] that experimented on inspection of object-oriented source code made a difference between defects that were local and non-local. Other papers that reported on inspection of source code used types that were related to where in the source code the defects was found, for example in small or large methods or wrong use of inheritance and so on.

6 Discussion

I will divide this chapter into two main parts, one for each of my research questions:

RQ1: *How is quality defined in a set of research papers on software inspection?*
RQ2: *How are defects defined and classified in a set of research papers on software inspection?*

6.1 Discussion regarding findings related to research question 1

As shown in section 5.2, only 4 of the 15 experiments analyzed discussed the impact software inspection can have on quality. 2 of the papers [biffl_00] and [biffl2_01] used the same terms for quality:

- Project Cost
- Number of defects

These papers both had good arguments for how project cost can be lowered by using inspection and how number of defects in a software engineering artifact can be lowered by the use of inspection. This was however not part of their main research and was mainly discussed in the motivation section of their papers.

Yet another paper [macdonald_et_al_98] used this term when talking about quality:

- Efficiency

This is just mentioned in a short sentence in the section about the experiment materials. They say that subjects should look at only functional defects and not defects related to other qualities such as *efficiency*.

One paper [zhang_et_al_99] used another term:

- Usability, with following sub attributes:
    - Speak the users' language
    - Consistency
    - Minimize the users' memory load and fatigue
    - Flexibility and efficiency of use
    - Use visually functional design
    - Design for easy navigation
    - Validation checks
    - Facilitate data entry
    - Provide sufficient guidance

This paper experimented on usability inspection of user interfaces and focused a great deal on usability. It is unquestionably the most quality focused experiment in my selection and the authors defines very well what they mean with the term usability. As it is quite a broad term it is even expanded to a list of nine sub attributes that they assign usability problems to.

So out of the four papers that took quality into account, two had a small discussion on how software inspection can improve quality, one mentioned it in an off sentence and the final one discussed it to a great extent. The other eleven papers did not discuss how software inspection affects quality at all, but only focused on how efficient the different software inspection methods were.

I think it is strange that software quality gets so little attention in these papers. One of the main goals for any software engineering undertaking *should* be to achieve as good quality as possible within the constraints that exists. Software inspection is a great tool for improving quality in many software engineering artifacts such as requirements documentation, source code, user interfaces, user documentation and so on. It is one of the best ways of detecting defects at an early stage of development and can help avoid many problems. In the organization I am employed in we among other use software inspection on critical pieces of source code, to ensure that no critical security loopholes or other critical bugs are deployed to a live environment, and it has proved to be very efficient at that for us.

It would be interesting to see formal experiments on exactly how quality and what forms of quality is impacted by software inspection, and for instance if any methods of software inspection is more suited to improving certain aspects of quality.

6.2 Discussion regarding findings related to research question 2

As shown in section 5.3, thirteen of the fifteen papers used some form of defect classification. Three of these papers exclusively classified defects by severity, seven exclusively classified by type and three classified by both severity and type.

6.2.1 Classification of defects by severity

Three of the papers that classified defects by severity used a scale of severities. One with three levels of severity [Johnson_et_al_97]:

- Low
- Medium
- High

One used a scale with four levels of severity [biffl2_01]:

- Trivial
- Minor
- Major
- Critical

The last one that used a scale used five levels of severity [zhang_et_al_99]:

- Not a usability problem
- Cosmetic problem only
- Minor usability problem
- Major usability problem
- Usability catastrophe

The first of these scales was an attribute in a computer tool for software inspection, and the meanings of the different severity levels was not discussed or defined in the paper. The second scale is discussed to some extent in the section about experiment materials and the meaning of the different severity levels is well defined. The last scale is very well defined and is part of a framework described by Nielsen et al [Nielsen_et_al_94] for usability inspection.

The other papers that had some mention of defect severity levels did only have brief discussions on how different inspection methods relate to defects of different severity levels, and did not use any scale or measure any severity levels for defects found.

I think that the severity level of defects should be a very natural thing to consider when researching software inspection. Almost all the papers measured how many defects a given inspection method found out of a known total. This is of course vital information, but it is probably just as interesting to know if an inspection method is good at finding defects of a certain severity level. Information about this would be helpful for project planners and managers in organizations to decide on which software inspection methods to use.

6.2.2 Classification of defects by type

Of the papers that classified defects by type, three ([Fusaro_et_al_97], [Miller_et_al_98] and [Porter_et_al_95]) actually used the same taxonomy which was originally proposed by Schneider et al [Schneider_et_al_92]. This taxonomy was made for inspection on requirements documentation and it is well defined and covers most types of defects related to requirements documentation. The other papers that reported on experiments on inspection of requirements documentation and used classification of defects by type used types similar to those in Schneider's taxonomy, but did not define them so well.

Those papers that was on software inspection of source code and classified defects by type ([Dunsmore_et_al_02], [Dunsmore_et_al2_00] and [Macdonald_et_al_98]) defined defect types based on where in the code they were located and what kind of coding errors was done. This seems like a reasonable thing to do, but the types could have been better defined in all of these papers.

Some of the papers mentions defect types ([Johnson_et_al_97] and [Porter_et_al_97_b]) but gives no effort to explain what they mean by these types or how they are defined.

In my selection of papers there has been some more effort to define defect types than defect severity levels, but much of the same can be said for the work on defect types as for the work on defect severity levels. Some papers have explained it rather well, some have scraped the surface and some have not given it any thought at all.

For all of the experiments the authors have been very eager to report the "raw efficiency" (how many defects from the total amount of defects existing in the inspection object that is found) of the software inspection methods they have researched, and this is perhaps the easiest metric to measure. But their research would be much more valuable in my eyes, and probably for decision makers and project managers in software organizations also, if they had reported in more detail what defect types and of which severity levels the different inspection methods is good at uncovering. If an inspection method excels at finding obvious defects that can be corrected by a project manager or system developer in a few minutes work time, but cannot be used for uncovering critical defects then it is not much worth.

7 Threats to validity

This section will discuss the most important threats to the validity of the results found in this thesis.

7.1 Selection of Articles

The review consisted of analyzing fifteen papers that reported on formal experiments on software inspection. Those papers was a subset of 37 papers on software inspection from an original 103 papers reporting on formal experiments on software engineering found for the study [Sjøberg_et_al_05]. The authors of that study had a very well defined strategy for finding as broad a selection as possible of controlled experiments on software engineering, but they report selection of articles as a possible threat to validity, so that concerns this thesis as well. Besides that I analyzed a "random" selection of the 37 possible papers where the randomization consisted of my supervisor sending me fifteen arbitrarily chosen papers. There is a risk that those fifteen papers is not representative of the original 37.

7.2 Data extraction

During the analysis I extracted data from fifteen papers. The data provided answers to various questions in the thesis. Considering the lack of related work to this thesis, guidelines and standards on how to extract data about quality metrics and defect classifications from research papers, it is possible that the data extraction may be tainted by subjectiveness from me. The most common way to address this validity issue is to have more reviewers to do the data extractions independently, which can be later compared and discussed. This was unfortunately not a possible option for this thesis.

I am also relatively inexperienced at conducting systematic reviews so it is possible that both my review protocol and extraction process is below professional scientific standard. A way to address this issue would have been to get help from a professional academic to write the review protocol and extract the data. The review protocol have been reviewed by my supervisor, but the data extraction could only be performed by me. To minimize the probability of extracting different data from the papers I have followed a pattern and a scheme throughout every analysis of each article. This can be seen if reading Appendix A or the review protocol.

8 Conclusion

8.1 Objective of research

The objective of the research was to study how quality is defined and how defects are classified and measured in a set of papers on formal experiments on software inspection.

I have done this by performing a systematic literature review on a selection of fifteen papers reporting on formal experiments on software inspection. The fifteen papers was a random subset of papers used in another research paper reporting on formal experiments on software engineering [sjøberg_et_al_05]. The data collected from the systematic review was used to answer my research questions:

RQ1: *How is quality defined in a set of research papers on software inspection?*
RQ2: *How are defects defined and classified in a set of research papers on software inspection?*

8.2 Findings

Population:

- *13 of the papers used students as subjects, 2 used professionals*
- *The number of subjects ranged from 30 to 177 for students and from 24 to 26 for professionals*

Quality

- *Only 4 of the 15 papers used any form of quality attributes*
- *The papers that used quality attributes did not define or explain why they used them very well*
- *In my opinion quality should get much more attention when researching software inspection than it has gotten in my selection of papers as software inspection is essentially a tool to enhance quality of a given software engineering artifact*

Defect classification

- *3 of 15 papers classified defects exclusively by severity*
- *7 of 15 papers classified defects exclusively by type*
- *3 of 15 papers classified defects by both severity and type*
- *2 of 15 papers did not give any effort to classifying defects*
- *3 papers used the same classification scheme for defect types*
- *Some of the papers used classification schemes that resembled each other, both for severity and type*
- *How much effort that were put into making solid classification schemes for defects varied wildly and some papers did not use classification of defects at all*

- *In my opinion a unified scheme for defect classification would serve the field of software inspection well.*

8.3 Future work

As my finding shows, there is very little attention given to software quality in my selection of papers. As software inspection is essentially a tool to enhance software quality I think more attention should be given to what kinds of software quality is benefited in what way from software inspection in future research on software inspection.

There is a little more attention given to classification of defects, but the classification schemes vary a lot, even for experiments that are more or less replications of each other. The lack of reporting how inspection methods perform in relation to defect types and severity levels makes the results hard to interpret and of low value both for other researchers and for decision makers in software organizations. I think it would be wise to make an effort to make unified defect classification schemes. They would of course need to differ according to what kind of software engineering artifact is inspected.

To validate the findings in this review it should be repeated with an even broader selection of papers. It would also be good to include more reviewers to do the data extraction in order to reduce the threats to validity.

Appendix A – Analysis of the selected articles

The empirical investigation of perspective based reading

Authors: Victor R. Basili, Scott Green, Oliver Laitenberger, Filippo Lanubile, Forrest Shull, Sivert Sørumgård, Marvin V. Zelkowitz

Setting: Two runs of the experiment with professionals from the NASA SEL environment, where the first run was a pilot experiment. No randomization of subjects as they had to accept whoever volunteered to participate. 12 subjects took part in the pilot run and 14 took part in the second run. Some of the subjects used perspective based reading technique from one of three perspectives, designer, user or tester and some subjects used what is referred to as the usual NASA technique.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

Defect classes are mentioned when talking about other reading techniques than those used in this experiment. This experiment did not look into what defect classes was detected, only the total sum of defects detected. However they say that defect classes will be the subject of future work.

A controlled experiment to assess the effectiveness of inspection meetings

Authors: Alessandro Bianchi, Filippo Lanubile, Giuseppe Visaggio

Setting: More than 100 students taking a two-semester software engineering course at the Univeristy of Bari. Students were randomly assigned to groups of either three or four subjects. All subjects first conducted an individual inspection, then a group inspection with their team.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

No mention or discussion of defect types or class in the paper. They used a premade lab package however, which might contain some information about this, but it is not looked further into or taken account for in the paper.

Using inspection data for defect estimation

Authors: Stefan Biffl

Setting: 169 undergraduate students divided into 31 teams ranging from 4 to 6 members doing inspection on an object representative of real world development specifications.

Findings regarding research question 1:

This quote summarizes what thoughts they have put into how software inspection relates to product quality.

"Product quality directly relates to project cost and schedule estimation; for example, undetected defects in a key work product – such as a requirements document might lead to time-consuming adjustments. Thus from the early project stages on, developers and project managers need to update their estimates of software project schedules and activities with feedback from the development process. A major aspect of this feedback is data on work-product quality levels (*number and type of deviations from specified quality goals*)."

Findings regarding research question 2:

This paper used no classification of defects.

Evaluating the accuracy of defect estimation models based on insepction data from two inspection cycles

Authors: Stefan Biffl, Willfried Grossman

Setting: 169 undergraduate students divided into 31 teams ranging from 4 to 6 members doing inspection on an object representative of real world development specifications.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

This quote from the paper describes how they classified defects. There were no further discussion on the defect classes.

"A defect in the requirements document was classified as missing, unnecessary, wrong, ambiguous, or inconsistent information."

Investigating the influence of inspector capability factors with four inspection techniques on inspection performance.

Authors: Stefan Biffl, Michael Halling

Setting: 177 undergraduate students from the University of Vienna. The experiment setup is not very well described, but checklist and scenario-based inspection techniques was used.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

There is a brief mention that during planning of inspection some should maybe focus on different defect types. Later it is mentioned that the scenario based reading technique instruct inspectors what defect class to uncover without explaining further what classes they mean. The paper found that a scenario-based reading technique helps inspectors find the most severe defects (defined as defects that cause considerably more rework later on, if not detected during inspection).

Investigating the cost-effectiveness of reinspections in software development.

Authors: Stefan Biffl, Bernd Freimut, Oliver Laitenberger

Setting: 169 students attending a two-semester university software development workshop on how to develop medium-sized software. They were split into 31 groups with 4 to 6 members. Teams were randomly assigned to using checklist or scenario-based inspection. A second cycle of inspection was carried out, a reinspection.

Findings regarding research question 1:

The paper says that the success of a software development project among other depend on the quality of the resulting product. This can be achieved by doing inspection, which can help with detection and removal of defects at an early stage. This saves rework efforts in later phases of the project. It is also said that the number of defects has something to do with quality level, "The objective of a second inspection cycle is to reduce the number of remaining defects to an acceptable quality level".

Findings regarding research question 2:

Defects are categorized by 4 severity levels 0, 1, 2 and 3, where 0 is a trivial and unimportant defect, 1 is a minor defect representing a local problem, 2 is a major defect which could be hard to find during development and could require multiple changes and finally 3 is a critical defect that may require reimplementation of the system. There is no discussion on how defects of the different severity levels impact the product quality; they are more focused on how they impact the cost of development.

Further investigations into the development and evaluation of reading techniques for object-oriented code inspection

Authors: Alastair Dunsmore, Marc Roper, Murray Wood

Setting: 69 3rd year students from the University of Strathclyde. They were split into three groups of 23 based on their ability. Each of the three groups focused on one review technique, either checklist, use-case based or a systematic technique.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

The paper discusses three different approaches to inspection, one of them being checklists. When introducing that they say that "a checklist is often limited to the detection of defects that belong to particular defect types". What these types are is not further discussed in the paper.

Later it is said that the checklists are divided into three sections that deal with issues on the class-level, method-level and finally method overriding issues. This does not seem to have any correspondence with the paragraph above, nor are any of these three sections deemed to be more or less important or critical than the others.

When the describing the experimental setup, they briefly discuss how the defects were introduced into the material. The only classification to be found here is that some of the defects have a delocalized nature. This because the paper studies inspection on object oriented work, which has a tendency to contain delocalized defects. Other information on the defects is not found in the paper, except for in the conclusion, where two of the inspection methods is said to be weak when encountering defects caused by missing lines of code.

Object-oriented inspection in the face of delocalisation

Authors: Alastair Dunsmore, Marc Roper, Murray Wood

Setting: 47 3rd year students from the University of Strathclyde. No grouping, only individual inspection. All groups used three review techniques, ad-hoc, checklist and scenario based both as individuals and as a team.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

Some natural defects (actual mistakes) and some artificially seeded in the code. Defects are classified where and how they appear in code, eg. their locality, whether they are in a small or large methods, wrong use of inheritance and so on. Defects are not classified by severity.

A replicated experiment to assess requirements inspection techniques

Authors: Pierfrancesco Fusaro, Filippo Lanubile, Giuseppe Visaggio

Setting: A replicated experiment, using 30 undergraduate students divided into 10 groups of three subjects each. The teams used a combination of ad-hoc, checklist and scenario-based inspection.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

The paper gives a definition of a detected defect as an "unstructured English description of what the subject believes to be incorrect, and it is required that an accurate description of the fault is supplied". Defects are classified according to a taxonomy originally proposed by Schneider et. al [schneider_et_al_92] and Basili and Weiss [basili_et_al_81].

Assessing software review meetings: A controlled experiment using CSRS

Authors: Philip M. Johnson, Danu Tjahjono

Setting: 72 undergraduate students from the University of Hawaii. They were divided into groups of 3, making a total of 24 groups. Each group was exposed to two treatments, real group review and nominal group review. A tool called CSRS (Collaborative Software Review System) was used to standardize the review process for both methods.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

In both rounds, the defects were mostly logic, computation, and data handling problems, such as missing or incorrect condition tests, forgotten cases or steps, and incorrect data access. Some of these defects were specific to the C/C++ languages, such as memory leaks. None of the defects, however, involved an incorrect specification. In fact, the participants were told beforehand that when the code did not conform to the specification, then the specification should be assumed correct, and the code was therefore incorrect.

This is not mentioned in the paper, but a screenshot of the software used for the reviews, shows that each defect must have a criticality level: High, medium, low or none. It must also have a confidence level of high, medium, low or none. It is not documented what the confidence level means, but probably it means how confident the reviewers are that the defect is actually a defect.

A comparison of tool-based and paper-based software inspection

Authors: Fraser Macdonald, James Miller

Setting: 43 3rd year students from the University of Strathclyde, split into two sections of 21 and 22 subjects. The two sections were balanced according to the subject's abilities. One of the sections used tool-based inspection while the other used paper-based inspection.

Findings regarding research question 1:

They do relate defects to certain quality attributes, like efficiency, but there is no complete discussion or taxonomy of this, just a brief mention in the description of the experiment setup.

Findings regarding research question 2:

They make a difference between defects that makes the code erroneous and defects relating to other qualities, like efficiency. Defects are also given a type and a level of how difficult they should be to discover, but this is not explained in detail.

Further experiences with scenarios and checklists

Authors: James Miller, Murray Wood, Marc Roper

Setting: 50 students divided into 16 groups. Each subject participated in two inspections using the same technique. The techniques applied in this experiment was checklist and scenario bases inspection

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

Defects are classified according to a taxonomy originally proposed by Schneider et. al [schneider_et_al_92] and Basili and Weiss [basili_et_al_81].

Comparing detection methods for software requirements inspections: a replicated experiment

Authors: Adam A. Porter, Lawrence G. Votta, Victor R. Basili

Setting: 48 graduate students divided into 16 groups of three subjects. Each team participated in two inspections using a combination of two out of three possible inspection methods, ad-hoc, checklist and scenario-based.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

Defects are classified according to a taxonomy originally proposed by Schneider et. al [schneider_et_al_92] and Basili and Weiss [basili_et_al_81].

Assessing software review meetings: Results of a comparative analysis of two experimental studies.

Authors: Adam A. Porter, Philp M. Johnson.

Setting: Comparative analysis of two experiments on software review.

Findings regarding research question 1:

This paper had no discussion on quality related to software inspection.

Findings regarding research question 2:

One of the hypotheses of the analysis says that some defects might be easier to find with group inspection than with individual inspection: "It remains an open question as to whether classes of defects exist whose detection costs and importance alone justify the use of meetings." Further the paper says that this is an important question for further research. If a class of defects that are more easily found with group inspection can be identified, then special purpose meetings for detecting these defects can designed and all other defects can be handled with individual inspection.

Perspective-based usability inspection: An empirical validation of efficacy

Authors: Zhijun Zhang, Victor Basili, Ben Shneiderman

Setting: Pilot experiment with 7 undergraduate students. Main experiment conducted with 24 professionals in a real organization. Each subject inspected two different artifacts in a random order, using one of three inspection perspectives, novice, expert or error handling.

Findings regarding research question 1:

The paper focuses on the quality attribute usability. Usability is a broad term and this is handled in the paper by breaking it down to more specific heuristics:

1. Speak the users' language
2. Consistency
3. Minimize the users' memory load and fatigue
4. Flexibility and efficiency of use
5. Use visually functional design
6. Design for easy navigation
7. Validation checks
8. Facilitate data entry
9. Provide sufficient guidance

Findings regarding research question 2:

This paper does not use the term defect, but rather usability problem. The usability problems are classified using Nielsen's rating scale [nielsen_93], which is a 5 level scale ranging from no usability problem to usability catastrophe.

Appendix B – Disclaimer on collaborative work in master thesis

This is a disclaimer on partly collaborative work in master thesis together with Mili Orucevic. The reason why we chose to do parts of the master thesis collaboratively is that the theses proposed to us by our supervisor, was quite similar. Both would require us to analyze a selection of papers describing controlled experiments, to find out how quality was described. We thought it would be an interesting addition to compare the findings of our theses. To be able to compare we had to have a similar understanding of quality and do the analysis as equally as possible. Therefore we decided to write the introduction, motivation and the parts about quality in general collaboratively. The review protocol for the papers was also made as similar as possible, so we could get data that was easily comparable.
So these sections of the theses are the same for both candidates:

- Introduction
- Software Quality

A section that is not completely similar but that was written in close collaboration is

- Research Method

Literature list

[asq_1] http://www.asq.org/learn-about-quality/history-of-quality/overview/overview.html

[asq_2] http://www.asq.org/learn-about-quality/iso-9000/overview/overview.html

[bar_88] Barfield , Owen. History in English Words. Great Barrington, MA: Inner Traditions/Lindisfarne Press, 1988. Reprint of original 1953 edition, Faber & Faber, London.

[basili_96_b] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S.Sørumgård, M.V. Zelkowitz, 1996:The empirical investigation of Perspective-Based Reading

[Basili_et_al_81] V.R. Basili and D.M. Weiss, "Evaluation of a software requirements document by analysis of change data", Proc. Fifth Int'l Conf. Software Eng., San Diego, Calif., Mar. 1981, pp. 314-323

[bianchi_et_al_01] A. Bianchi, F. Lanubile, G. Visaggio 2001: Proceedings of the Seventh International Software Metrics Symposium (METRICS .01): A Controlled Experiment to Assess the Effectiveness of Inspection Meetings

[biffl_00] S. Biffl 2000: IEEE SOFTWARE November/December 2000: Using Inspection Data for Defect Estimation

[biffl_01]: S. Biffl, W. Grossmann 2001 IEEE: Evaluating the Accuracy of Defect Estimation Models Based on Inspection Data From Two Inspection Cycles

[biffl_et_al] S. Biffl, M. Halling 2002: Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS.02): Investigating the Influence of Inspector Capability Factors with Four Inspection Techniques on Inspection Performance

[biffl2_01] S. Biffl, B. Freimut, O. Laitenberger 2001 IEEE: Investigating the Cost-Effectivenessof Reinspections in Software Development

[cro_79] P.B. Crosby, Quality is Free, McGraw-Hill, 1979

[dunsmore et al 02] A. Dunsmore, M. Roper, M. Wood 2002: ICSE' 02, May 19-25, 2002, Orlando, Florida, USA: Further Investigations into the Development and Evaluation of Reading Techniques for Object-Oriented Code Inspection

[dunsmore_et_al2_00] A. Dunsmore, M. Roper, M. Wood 2000: ICSE 2000, Limerick, Ireland: Object-Oriented Inspection in the Face of Delocalisation

[eps_91] Steven A. Epstein: Wage & Labor Guilds in Medieval Europe

[Fagan_76] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, 15(3):182–211, 1976.

[fusaro_et_al_97] P. Fusaro, F. Lanubile, G. Visaggio 1997: Empirical Software Engineering, 2,19-57 (1997):A Replicated Experiment to Assess Requirements Inspection Techniques

[hum89] W. Humphrey, "Managing the Software Process", Addison-Wesley 1989

[Johnson_et_al_97] P.M. Johnson, D. Tjahjono 1997: ICSE 97 Boston MA USA: Assessing software review meetings: A controlled experimental study using CSRS

[jur_gry_70] Juran and Gryna: Quality Planning and Analysis: From product development through use, McGraw-Hill, 1970

[kan_04] Stephen H. Kan: Metrics and Models in Software Quality Engineering, pp. 1-4, Published 2002 Addison-Wesley

[kitchenham_04] Kitchenham B.A., Procedures for Performing Systematic Reviews, Keele University,Technical report TR/SE-0401 and NICTA Technical Report 0400011T.1, 2004

[Macdonald_et_al_98] F. Macdonald, J. Miller 1998: Empirical Software Engineering, 3, 233–253 (1998): A Comparison of Tool-Based and Paper-Based Software Inspection

[miller_et_al_98] J. Miller, M. Wood, M. Roper 1998: Empirical Software Engineering, 3, 37–64 (1998): Further Experiences with Scenarios and Checklists

[nett_0607] http://www.nettavisen.no/bil/article1097164.ece

[Nielsen_93] Nielsen, J. 1993. Usability Engineering. San Diego: Academic Press, Inc.

[porter_et_al_95] A.A. Porter, L.G. Votta, V.R. Basili 1995: IEEE TRANSACTIONS ON SOFTWARE ENGiNEERING, VOL. 21, NO. 6, JUNE 1995 : Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment

[porter_et_al_97_b] A.A. Porter, P.M. Johnson 1997: IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 23, NO. 3, MARCH 1997: Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies

[scalet_etal_00] Scalet et al, 2000: ISO/IEC 9126 and 14598 integration aspects: A Brazilian viewpoint. The Second World Congress on Software Quality, Yokohama, Japan, 2000.

[Schneider_et_al_92] G.M Schneider, J. Martin, and W.T. Tsai, "An experimental study of fault detection in user requirements", ACM Trans. Software Eng. and Methodology, vol. 1, no. 2, pp 188-204, Apr. 1992.

[SEG_97] SEG and Department of Computer Science, University of Durham, "Guidelines for performing Systematic Literature Reviews in Software Engineering", 9.July 2007

[sjøberg_et_al_05] D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N-K. Liborg and A.C. Rekdal, "A survey of controlled experiments in Software Engineering", IEEE Trans. on soft. eng. vol. 31, no. 9 Sept. 2005

[sqmap] Software quality models and philosophies, http://www.bth.se/tek/besq.nsf/(WebFiles)/CF1C3230DB425EDCC125706900317C44/$FILE/chapter_1.pdf

[zhang_et_al_99] Z. Zhang, V.R. Basili, B. Shneiderman 1999: Empirical Software Engineering, 4, 43–69 (1999): Perspective-based Usability Inspection: An Empirical Validation of Efficacy