UNIVERSITY OF OSLO
Department of informatics

Quality definitions and measurements in software engineering experiments on pair programming

Master thesis
<60 credits>

Mili Oručević

<30. April 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 what the term quality means. To understand it, we have to have a clear definition of what quality is, which is easier said than done. Even though 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.

This thesis studies the concept of the term quality used in a set of selected experiments on the effectiveness of pair programming. I will present an overview that characterizes what authors and researchers call quality in empirical software engineering. The objective of the investigation is to get an overview of how quality is defined, described and measured in experiments which study the effect of pair programming, and how to improve software quality.

The findings show that there are great differences and variances in software engineering studies regarding pair programming. Not only is there a variance in, as expected the outcomes, but there is a great variance also in number and type of subjects used, quality definitions, metrics used, and last but not least claims based on different measurements.

Surprisingly many authors were biased in the articles analyzed in this thesis. It seemed that the only agenda they had was to persuade readers to share their opinion and view on a matter, instead of presenting the issue without shining the light favorably on one side of the case. In other words present both cons and pros and let the reader take a standing point in the issue.

None of the authors explained what their criteria were for including or excluding the metrics they used in the studies. The authors claiming that a certain programming technique brings much more quality to the end product do not state explicitly or vaguely what they mean with the phrase “better quality”. It is no wonder that researchers and professionals in the industry are confused when it comes to use of the term quality.

To make the use of quality metrics even more difficult to understand, the authors tend to use quality metrics of different types. It is shown that authors, who are advocates of pair programming, use more subjective measurements to claim that pair programming is “better” than solo programming, than authors who are neutral to the subject.

This thesis show that throughout the articles regarding pair programming efficiency the authors inconsistently use quality definitions and measurements and later use these to claim their results to be “valid”. This inconsistency seems to take overhand and if not viewed upon with more critical eyes it could lead to a step back in maturing the software engineering community and research. When leading researchers fail to define, measure, and use the quality metrics correctly, it should not come as a surprise that students and other researchers mix up and treat these issues inconsistently. It all seems simple, but in reality it is extremely difficult.

These findings should act as a wakeup call to the software engineering community, and to the research field of pair programming. Everything points towards that the software engineering research community needs a lot more maturing, and a lot of more experiments and other empirical studies to

grow towards a reliable source of knowledge. Most of the studies done up to today are premature in many aspects. From not agreeing on common and universal standards, lack of knowledge on how to conduct unbiased experiments, to blindly accepting findings and claims done by authors who have a foot planted inside the software engineering research industry.

Acknowledgements

First of all, I would like to thank my supervisor, Dag Sjøberg, for his incredible and priceless support, contributions, guidance, encouragement, and discussions. I am sincerely grateful for his continuous inspirations during the work with this thesis. Great thank to my good friend and fellow student Morten H. Bakken for useful comments, and hours of discussion. Thanks to the students and employees at Simula Research Laboratory for making a nice work environment during the last year.

Last but not least, thanks to my family, loved ones and friends for their support and encouragement throughout this period.

Oslo, April 2009
Mili Oručević

Contents

List of tables

List of figures

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 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 with each other. This thesis investigates how the term quality is used within experiments that describe the pair programming technique. This specific development technique is becoming more and more popular, but to what extent should the program managers promote this technique?

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 Pair Programming

Pair programming, by definition, is a programming technique in which two programmers work together at one computer on the same task [williams_03]. The person typing is called the driver, and the other partner is called the navigator. Both partners have their own responsibilities; the driver is in charge of producing the code, while the navigator's tasks are more strategic, such as looking for errors, thinking about the overall structure of the code, finding information when necessary, and being an ever-ready brainstorming partner to the driver [hulkko_abra_05].

Pair programming is one of the key practices in eXtreme Programming (XP). It was incorporated in XP, because it is argued to increase project members' productivity and satisfaction while improving communication and software quality [hulkko_abra_05 ref. Beck].

In today's literature many benefits of pair programming have been proposed, such as increased productivity, improved software quality, better confidence among the programmers, as well as satisfaction, more readable programs, etc. However there have been also findings of negative outcome as well. Pair programming is criticized over increasing effort expenditure, overall personnel cost, and bringing out conflicts and personality clashes among developers. The empirical evidences behind these claims are scattered and unorganized, so it is hard to draw a conclusion in one way or another, to favor one of the sides. This is also one of the reasons why pair programming has not been adopted by the industry, there is no firm evidence of weather it is better or worse than solo programming.

1.3 Problem formulation

This thesis studies the concept of quality used in a set of selected experiments on the effectiveness of pair programming. I present an overview that characterizes what authors and researchers call quality in empirical software engineering. On the basis of this overview, other authors and researchers may decide further research for improving the use of term quality.

The objective of the investigation is to get an overview of how quality is defined, described and measured in experiments which study the effect of pair programming.

In order to address these issues, I have analyzed a set of articles which study the effect of pair programming. The set consist of 15 articles, which are selected from a larger set (214 articles). In Chapter 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 following research questions:

> ***RQ1:*** *How is quality defined in a set of articles describing the effect of pair programming technique?*
> ***RQ2:*** *How is quality measured in this set of articles?*

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. Chapter 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. Chapter 3 "Related Work" sums up relevant and related work to this thesis done by different authors. Chapter 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. Chapter 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. Chapter 6 "Discussion" discusses the findings done in the previous section, what authors call quality, their affiliations and more. Chapter 7 "Threats to Validity" addresses the issue regarding validity in this thesis, and in Chapter 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

Historians suggest that it is 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

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].

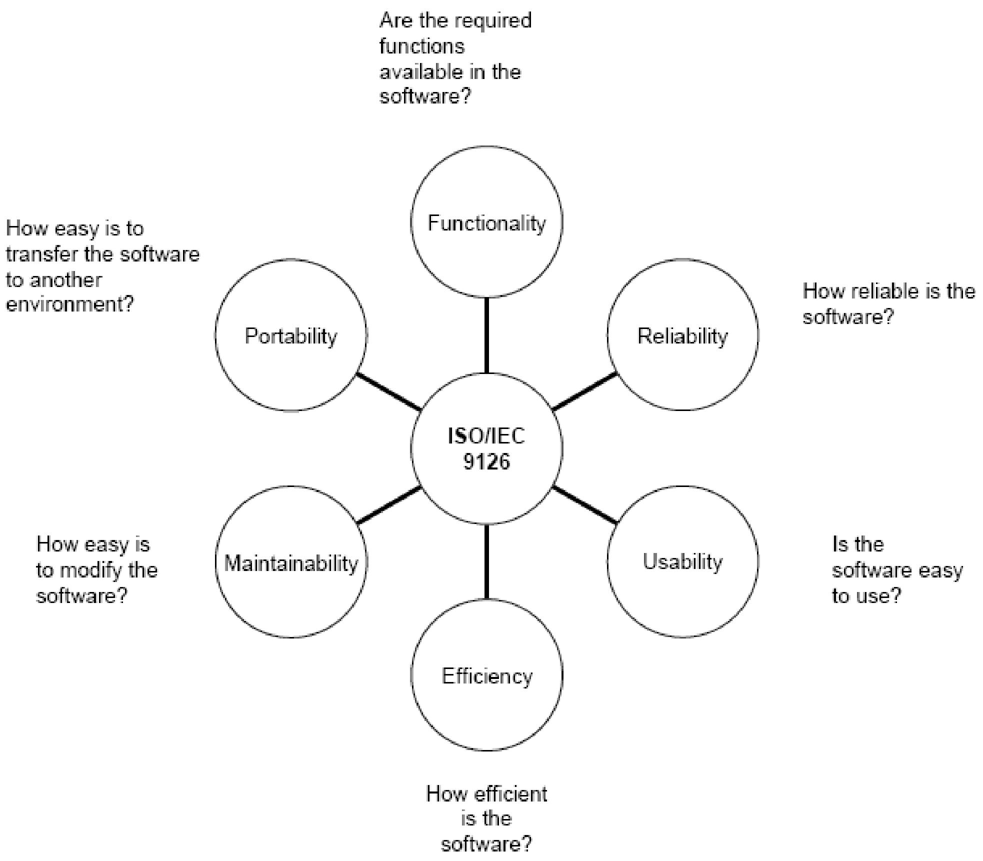
- **Reliability**
  A set of attributes that relate to the capability of software to maintain its level of performance understated conditions for a stated period of time [sqmap].

- **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].



Are the required functions available in the software?
Functionality
How easy is to transfer the software to another environment?
Portability
Reliability
How reliable is the software?
ISO/IEC 9126
How easy is to modify the software?
Maintainability
Usability
Is the software easy to use?
Efficiency
How efficient is the software?

Figure 1: Main factors in ISO 9126

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

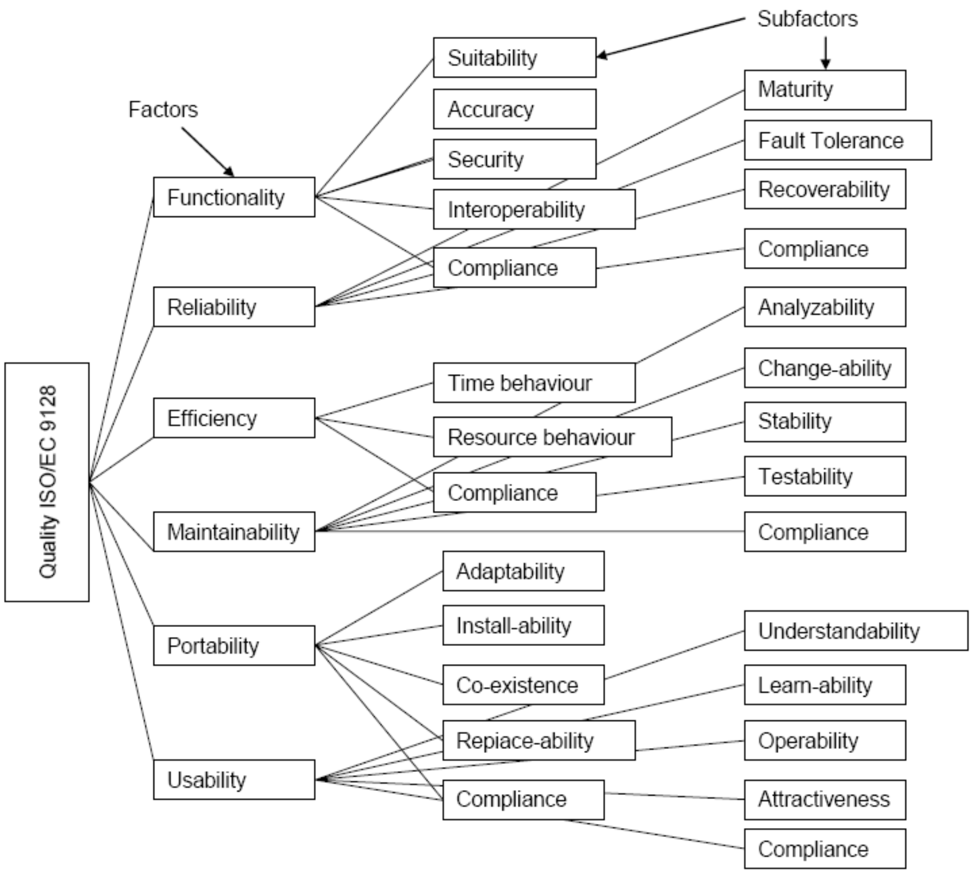


Figure 2: Main and sub-factors in ISO9126

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

3 Related work

So far there are no known studies that focus on how quality is defined and measured in software engineering experiments regarding the effectiveness of pair programming. In fact no studies exist regarding investigation of how quality is defined and measured using any programming technique.
Only two studies exist regarding pair programming with focus on quality parameters, these are; [dybå_etal_07] which analyzes the same 15 articles that are analyzed in this thesis, and [hannay_etal_09] which is a meta-analysis of [dybå_etal_07].

This thesis is the one of the first exploring this field, setting the focus on what does it mean when authors claim that a programming technique produces software with higher quality. Are the quality metrics defined, is it explained why some quality metrics are used, and not others? These are some of the issues that will be discussed in later sections.

There are many studies that concern the effect of pair programming, especially versus solo programming. Studies on other programming techniques can be found as well, where pair programming is just a part of the study, i.e. pair programming is compared to the researched technique. Before pair programming became widespread as an Extreme Programming practice, Wilson et al. [wilson_etal_93] investigated collaborative programming in an academic environment. They found evidence that collaboration in pairs reduced problem-solving efforts, enhanced confidence in the solution and provided a better enjoyment of the process. Nosek [nosek_98] confirmed the results in a controlled experiment, involving experienced developers, and he found that coupled developers spent 41% less time than individuals, and produced better codes and algorithms. It is important to highlight that collaborative programming is not the same as pair programming. The former refers to a group of two or more people involved in coding, without adopting a specific working protocol; the latter is a practice involving only two people and with a precise protocol which prescribes to continuously overlapping reviews and the creation of artifacts.

Williams et al. [williams_etal_00] carried out one of the most well known experiments on pair programming, which involved the participation of senior software engineering students. By working in pairs, the subjects decreased development time by 40–50%, and passed more of the automated test cases; moreover, the results of the pairs varied less in comparison to those of the individual programmers. Several other investigations have highlighted the benefits of pair programming ([lui_chan_03], [mcdowell_etal_02], [williams_03]): effort is reduced and quality is improved.

However, these results were not confirmed by two other experiments: the first experiment executed at the Poznan University (Nawrocki and Wojciechowski, [nawrocki_woj_01]) demonstrated how pair programming was able to reduce rework, but it did not significantly reduce development time, and the second conducted by Heiberg et al. [heiberg_etal_03] demonstrated how pair programming was neither more nor less productive than solo-programming. The relationship between pair programming and the geographic distribution of teams was explored by Baheti et al. [baheti_etal_02]: distributed pair programming was comparable with co-located pair programming and fostered teamwork and communication within virtual teams. Other investigations have highlighted further benefits of pair programming, such as: fostering knowledge transfer [williams_kessler_00], in particular, leveraging of tacit knowledge, increasing job satisfaction [succi_etal_02] and enforcing student learning ([mcdowell_etal_02]; [xu_rajlich_06]). Conversely, few studies focus on pair designing. Al-Kilidar [al-kilidar_etal_05] carried out an experiment in order to compare the quality obtained by solo and pair work in intermediate design products. The experiment showed that pair design quality was higher than solo design quality in terms of the ISO 9126 sub-characteristics: functionality, usability, portability and maintenance compliance. Müller [muller_06] presented the results of a preliminary study that analyzed the cost of implementation with pair and solo design; the results of this study suggested that no difference exists, assuming that the programs have similar levels of correctness. The authors also concluded that the probability of building a wrong solution into the design phase might be much lower for a pair than for a single programmer.

Despite the growing interest, practitioners can face difficulties to making informed decisions about whether or not to adopt pair programming, because there is little objective evidence of actual advantages in industrial environments. Most published studies are based in Universities and involve students as their subjects. Hulkko and Abrahamsson [hulkko_abra_05] state that ‘‘the current body of knowledge in this area is scattered and unorganized. Reviews show that most of the results have been obtained from experimental studies in university settings. Few, if any, empirical studies exist, where pair programming has been systematically under scrutiny in real software development projects’’. More experimentation in industry is needed in order to build a solid body of knowledge about the usefulness of this technique as opposed to traditional solo programming development.

When it comes to software quality, more and more studies are found regarding this topic, as assuring software quality becomes a more important part of the development process. These studies usually investigate whether there are a unified term for quality, a unified measurement etc.

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

4.2.1 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_07]. Also, with a strict structure of how to analyze the articles, it will be easier to compare the articles to each other.

4.2.2 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

4.2.3 Research questions

RQ1: How is quality defined in a set of articles describing the effect of pair programming technique?
RQ2: How is quality measured in this set of articles?

4.3 How articles were selected and analyzed

The selection of the articles is done in a prior study [dybå_etal_07], which investigated the effectiveness of pair programming and compared with solo programming. Since the selection has been done without my involvement I will just replicate the method used in Dybå's paper [dybå_etal_07].

Dybå [dybå_etal_07] followed general procedures for performing systematic reviews, as suggested by Kitchenham in his paper [kitchenham_04], which are based largely on standard meta-analytic techniques.

4.3.1 Inclusion and exclusion criteria

The authors of [dybå_etal_07] examined all published English-language studies of pair programming in which a comparison was made (a) between isolated pairs and individuals, or (b) in a team context. Studies that examined pair programming without comparing it with an alternative were excluded.

4.3.2 Data sources and search strategy

The authors searched the ACM Digital Library, Compendex, IEEE Xplore, and ISI Web of Science with the following basic search string: "pair programming" OR "collaborative programming." In addition, the authors hand-searched all volumes of the following thematic conferences proceedings for research papers: XP, XP/Agile Universe, and Agile Development Conference.

4.3.3 Study identification and selection

The identification and selection process consisted of three major stages. At stage 1, the authors applied the search terms to the titles, abstracts and keywords of the articles in the identified electronic databases and conference proceedings. Excluded from the search were editorials, prefaces, article summaries, interviews, news items, correspondence, discussions, comments, reader's letters, and summaries of tutorials, workshops, panels, and poster sessions. This search strategy resulted in a total of 214 unique citations.

At stage 2, two of the authors Hannay and Dybå, went through the titles and abstracts of all studies resulting from stage 1 for relevance to the review. If it was unclear from the title, abstract and keywords whether a study conformed to our inclusion criteria, it was included for a detailed review. At this stage, we included all studies that indicated some form of comparison of pair programming with an alternative. This screening process resulted in 52 citations that were passed on to the next stage.

At stage 3, the full text of all 52 citations from stage 2 were retrieved and reviewed by the same two authors. All studies that compared pair programming with an alternative, either in isolation or within a team context, were included. This resulted in 19 included articles. Four of these did not report enough information to compute standardized effect sizes and were excluded. Thus, 15 studies (all experiments) met the inclusion criteria and were included in the review.

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 to extract following data:

- *General information*
    - Date of data extraction
    - Title, author, journal, publication details
- *Quality information*
    - How is quality defined on effect of pair programming technique?
    - How is quality measured?
    - Quality description and measurement (which quality attributes were used, which not, weather bias occurred, ex. "*Only the quality attributes that provided positive results were included".*)
- *Other Specific information*
    - Population selection
    - Outcomes
    - Possible motivational biases in estimation

The general information is to present to the reader title of the article, who wrote them, who published them, and who financed the investigation because of bias. If a company "X" finances an investigation to perhaps compare their product with rival product, the results often end up on company "X"'s side. When it comes to quality definitions and measurements, if company "X" knows that their product scores low on maintainability, they will perhaps exclude this quality definition from the research, to end up with a better overall result.

The main goal of this study is to find out how quality is defined, and how it is measured in the different articles, also if the definitions and measurements are similar to each other. For each article I will try to extract the following information:

- What quality attributes is chosen to represent the outcome?
- How quality attributes are defined?
- How quality attributes are measured?
- Whether there is a bias in the selection of the quality attributes

In the other specific information I will look into who the population was in the research papers, this is because there is a gap between academy and the market. Perhaps some of the academics don't emphasize on a certain quality aspect when they are writing programs, where professionals do. The outcome of the articles studied is not so relevant to this paper, if the studies are not focused on quality directly. Although it can be relevant, as mentioned earlier, if one wants to manipulate the findings to enlighten a certain method, or technique, one can deliberately exclude some quality attributes if one knows that the specific method or technique scores poorly. At the end I will mention possible biases that occur, if any found.

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.

The first section summarizes the findings done by the analysis. At first I will presents an overview of all the articles in general, table 2 and the first paragraphs. This means 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 findings per article, which means what attribute is considered to be a quality attribute by each article. Afterward I will provide information on which article mentions which quality attribute, and how many articles mention the same quality attribute.

5.1 Summary of the studies

Table 2: Summary of all studies

Study	**Subjects**	**Total amount of subjects**	**Study setting**
arisholm_etal_07	Professionals	295	10 experimental sessions with individuals over 3 months and 17 sessions with pairs over 5 months (each of 1 day duration, with different subjects). Modified 2 systems of about 200-300 Java LOC each.
baheti_etal_02	Students	98	Teams had 5 weeks to complete a curricular OO programming project. Distinct projects per team.
canfora_etal_05	Students	24	2 applications each with 2 tasks (run1 and run2).
canfora_etal_06	Professionals	18	Study session and 2 runs (totaling 390 minutes) involving 4 maintenance tasks (grouped in 2 assignments) to modify design documents (use case and class diagrams).
heiberg_etal_03	Students	84	4 sessions over 4 weeks involving 2 programming tasks to implement a component for a larger "gamer" system.
madeyski_06	Students	188	8 laboratory sessions involving one initial programming task in a finance accounting system (27 user stories).
muller_04	Students	20	2 programming tasks (Polynomial and Shuffle-Puzzle)
muller_05	Students	38	2 runs of 1 programming session each on 2 initial programming tasks (Polynomial and Shuffle-Puzzle) producing about 150 LOC.
nawrocki_woj_01	Students	15	4 lab sessions over a winter semester, as part of a university course. Wrote 4 C/C++ programs ranging from 150-400 LOC.
nosek_98	Professionals	15	45 minutes to solve 1 programming task (script for checking database consistency).
phongl_boehm_06	Students	95	12 weeks to complete 4 phases of development + inspection.
rostaher_her_02	Professionals	16	6 small user stories filling 1 day.
vanhanen_lass_05	Students	16	9-week student project in which each subject spent a total of 100 hours (400 hours per team of four). 1500-4000 LOC were written.
williams_etal_00	Students	41	6-week course where the students had to deliver 4 programming assignments.
xu_rajlich_06	Students	12	2 sessions with pairs and 1 session with individuals. 1 initial programming task producing around 200-300 LOC.

5.1.1 Characteristics of the 15 studies

Of the 15 studies, 10 were from Europe and 5 from North America. In 11 of the studies the subjects were students, while in the other 4 professionals were used. 11 of the studies compared the effectiveness of isolated pairs vs. isolated individuals, and only four studies made the comparison within a team context. All studies used programming tasks as the basis for comparison. In addition, Madeyski [madeyski_06] included TDD (test driven development), Muller [muller_04] and Phongl and Boehm [phongl_boehm_06] included inspections, and Muller [muller_05] included design tasks. The number of subjects varied from 12 to 295, with a median of 24.

5.1.2 Population selection

In figure 3 we can see that subjects in 11 of the studies consisted of students, while the 4 remaining studies used professionals as subjects. Many claim that students are not a representative population group, due to many reasons, but some of the authors use students because they are cheaper, more flexible, open minded, and also Williams [williams_03] argues that they are the new working force in the industry and therefore are a representative group in same matter as the professionals.

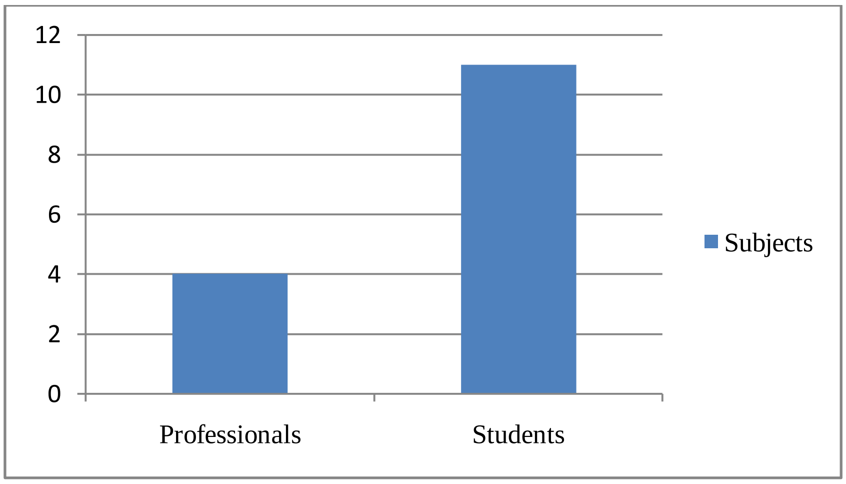


Figure 3: Number of different subjects participating in the studies

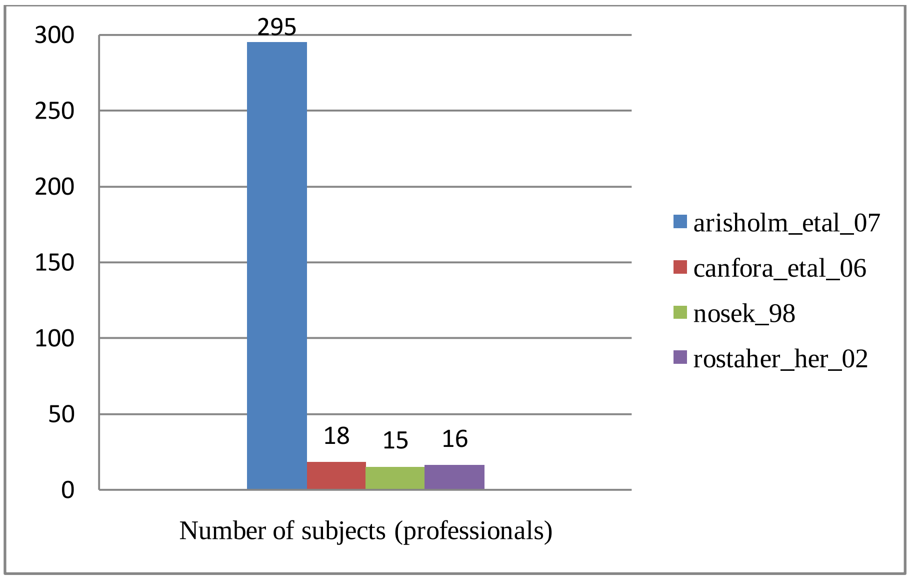


Figure 4: Number of subjects (professionals) used in the studies

Out of the 4 studies done with professionals as subjects, only one of them had a significant number of subjects. The other 3 had clearly too few subjects participating (see figure 4).

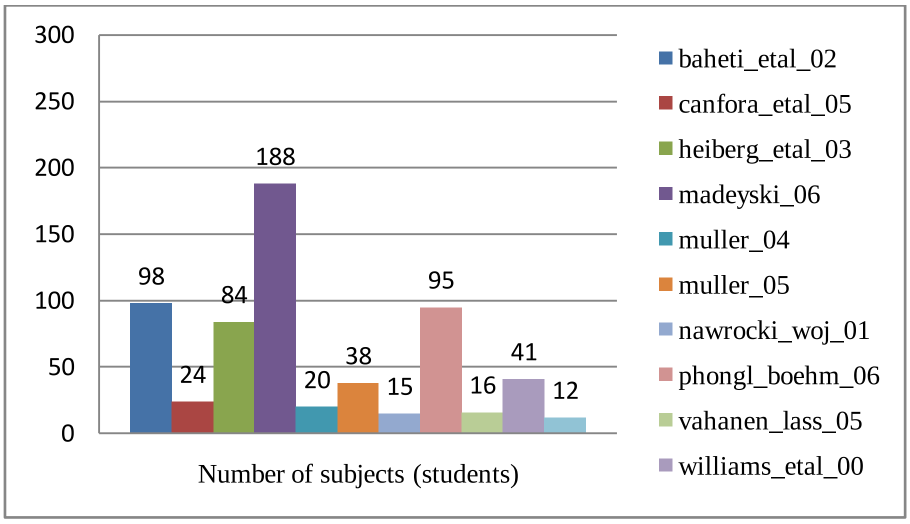


Figure 5: Number of subjects (students) used in the studies

Most of the studies use quite a small number of subjects in their experiment (see figure 5), even though they mention it in the conclusion of their papers. My opinion is that there is not a issue due to the small number of subjects, but what creates an issue is the combination of small number of subjects combined with that those subjects are students. There is quite a big gap between students' knowledge and skills in programming, on this level. I suppose this is an issue for professionals also, where if one is to investigate something in a company they will certainly not allow you to use their best employees. One more thing worth mentioning, most of the academic studies were designed such that those students who are especially interested in this subject participated, which does not give a unified selection of subjects. It is a complex issue altogether, and the studies which were investigated in this thesis use different criteria for choosing their subjects.

5.1.3 Outcome of the studies

The outcome of the studies analyzed has quite little relevance for the research question of this study. The only reason that this could affect the research question is that the authors of the studies analyzed deliberately ignored some metrics because they "knew" certain metrics would present their case in a negative matter. This is almost impossible to detect due to the lack of explanation by the authors why some quality metrics have been included in the studies and why some have not. I will just present some main arguments for why pair programming could be better than solo programming, as well as some counter arguments.

One of the main arguments for compensating the increased overall project costs due to higher effort expenditure of pair programming is improved quality of the resulting software [williams_03]. Proposed reason for the quality improvements include the continuous review preformed by the navigator, which is claimed to outperform traditional reviews in defect removal speed [williams_03], enhance programmers' defect prevention skills, and pair pressure which according to Beck [beck_99], encourages better adherence to process conventions like coding standards and use of refactoring. These are some of the arguments that claim why pair programming is better than regular solo programming. On the other hand Nawrocki [nawrocki_woj_01] reports differently than both Williams

[williams_etal_00] who claim that pair programming reduce development time by 50 %, and Nosek [nosek_98] who support that claim by showing similar results in his own study (29 % reduction in development time). Nawrocki [nawrocki_woj_01] claims that pair programming is less efficient than Williams [williams_etal_00] and Nosek [nosek_98] report. According to Nawrocki's study pairs appeared less efficient than previously reported, and showed that solo programmers using XP techniques used same amount of time as pairs.

While most of these findings point to a positive direction, the generalizability and significance of these findings remains questionable. One reason for this is the fact that often the metrics used for describing quality have not been either defined in detail in the studies, or lack the connection to the quality attribute they should be presenting.

5.1.4 Possible motivational biases in the studies

There are several possible motivational biases in the studies analyzed, one of the main issue is that the authors who writes the articles seem to have a bias towards one of the programming techniques in advance, before they start to undertake the experiment. Some of the authors are advocates for the pair programming cause, and this comes out clearly in their reporting, where they try to magnify their results to present their cause as positive as possible. On the other hand there are a few anti-pair programming authors who try to counter the pair programming technique. The reason why this is mentioned is that this affects the results in the studies done. The advocates of pair programming choose to use only metrics that present pair programming in a positive light, while the anti-pair programming authors do the opposite. This is difficult to prove because none of the authors explain why they choose to include metrics of one kind, and exclude other metrics. Since there is no neutral organization that can conduct and investigate these results, the only thing remaining is to use common sense and try to see the big picture in this matter.

Other biases that occur are i.e. in [xu_rajlich_06] the authors try to investigate whether pair programming is suitable for game development industry, and their assignment which is to keep score of bowling results does not resemble a modern computer game. A modern computer game is almost all about the graphics and the problems which this issue can cause is not address by the author. Also students were used in this experiment, which is a bit odd population group to test this. If one only looks at the results this can be seen out of context, and distort the readers view on this matter.
Heiberg [heiberg_etal_03] mention and compares ”traditional teamwork” with pair programming, without ever explaining what traditional teamwork means to him. Nosek [nosek_98] uses plainly subjective measurements except for time used, to claim that pair programming is clearly superior to solo programming. With only subjective measurements one can claim a lot of things without them being true or false. It is impossible to investigate this. In other papers the authors act as a mentor in the class which is participating the experiment, this can also mean that the author can affect the subjects in a way or another and distort the results. In [willams_etal_00] 2 of the co-authors are founders of the XP practice and little suggest that this article is un-biased. I clearly doubt that an inventor of the XP practice wants to present his cause with some poor results.

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, every author used some form of quality metrics, either process quality metrics or product 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 pr. article

Table 3: Quality metric pr. article

Study	Quality metrics	Defined and measured
arisholm_etal_07	Duration Effort Correctness	Elapsed time in minutes to complete change task Total change effort in person minutes per subject A binary functional correctness score with value “1” if all change tasks were implemented correctly
baheti_etal_02	Productivity ”Quality” Communication within the team Cooperation within the team	Line of code per hour Average grade obtained by the group Subjective measurement Subjective measurement
canfora_etal_05	Effort Predictability	Time used on a task Standard deviation derived from time used
canfora_etal_06	Effort ”Quality” Predictability	Time used on a task Subjective score ratio given for each task by the author and two independent evaluators Analyses according to standard deviation numbers
heiberg_etal_03	”Quality of the solution”	Passed test cases
madeyski_06	Afferent coupling Efferent coupling Instability Abstractness Normalized distance from main sequence	All these metrics are measured using an internal tool called AOPmetric [aopmetric]. The metrics are explained in detail in appendix A.

muller_04	Reliability
	Cost
		Test cases passed
		Time used,
		Pairs = 2x (Read phase + Implementation phase + QA phase)
		Review = Read phase+ Implementation phase + Review phase + QA

muller_05	Reliability Cost	Test cases passed Time used, Pairs = 2x (Read phase + Implementation phase + QA phase) Review = Read phase+ Implementation phase + Review phase + QA
nawrocki_woj_01	Total development time Programming efficiency Software process efficiency indicator	Time used for each program Line of code pr hour Number of retransmissions after acceptance test
nosek_98	Readability Functionality Time used Confidence	Subjective measurement Subjective measurement Time used in minutes Subjective measurement

	Enjoyment	Subjective measurement
phongl_boehm_06	Total Development Cost Distribution of cost Defects found Un-passed test cases Project score	Time in man hours Percentage of total cost Defects found by teacher assistant after each phase Numbers of test cases unpassed Overall project score measured from all previous measurements combined
rostaher_her_02	Relative time spent Relative time spent for specific programming activity Switching between driver/navigator role	Time spent (hours, minutes) Percentage of the total time Count of how many times subjects switched
vanhanen_lass_05	Productivity Defect rate Design quality Knowledge transfer Enjoyment of work	Amount of work/effort spent Defects counted pre-delivery and post-delivery Non-comment line of code per method Subjective measurement Subjective measurement
williams_etal_00	Productivity Defect rate (”Development/Software quality”) Effort Satisfaction	Time used to complete assignments Test case passed Time used multiplied by number of developers Subjective measurement
xu_rajlich	Program length Efficiency Cohesion More meaningful variable names Robustness Total time cost	Line of code counted Line of code written pr hour Class members generated Subjective measurement Black box test cases passed Time used pr subject

Out of the 15 studies the definitions and measurements of quality divert from one article to another. The main reason for this is that none of the authors use any standardized quality metric but rather come up with their own definition and meaning. None of the authors elaborate on this matter, and they neither explain why they choose to include some metrics in their research and why exclude others.

Table 4 shows what kind of quality metrics are used in the set of selected articles. Only metrics that were used in 2 or more studies are shown. The reason for this is to eliminate the quality metrics that are only used once because it is difficult to compare something that is used in one article but not in another. This can also lead to biased results because some of the articles are written by same authors/co-authors, and if they investigated the same “effect” of pair programming, the quality metric they used will be counted several times. Also if proper selection of metrics is done by just one author and none of the others, these metrics are not elaborated on more.

Table 4:Amount of quality metrics found in the articles

Quality metric	**Definition**	**No. articles**	**Other names for the same metric**
Duration[1]	Time used to complete the task	8	Effort[3,4], Total Development Time[9], time used[10], Total Development Cost[11], Relative time spent[12], Productivity[14]
Effort[1,14] **/ Cost**[7,8]	Effort/Cost in x amount of time per subject	6	Productivity[13], Total time cost[15]
Correctness[1]	Binary functional correctness score	2	Project score[11]
Productivity[2]	Line of code per hour	3	Program efficiency[9], Efficiency[15]
Predictability[3,4]	Standard deviation derived from time used	2	
Reliability[7,8]	Test cases passed	6	"Quality of the solution"[5], Un-passed test cases[11], "Development Quality"[14], Robustness[15]
Defects found[11]	Number of defects found	2	Defect rate[13]
Readability[10]	Subjective measurement	2	More meaningful variable names[15]
Enjoyment[10,13]	Subjective measurement	2	
Distribution of cost[11]	Measured how much time was used in different phases of the development cycle	2	Relative time spent for specific programming activity[12]
Program length[15]	Lines of code counted	1	
"Quality"[2]	Average grade obtain by the group	1	
Communication within the team[2]	Subjective measurement	1	
Cooperation within the team[2]	Subjective measurement	1	
Afferent coupling[6]	The number of classes outside the package that depend upon classes within the package. Measured using [aopmetric].	1	
Efferent coupling[6]	The number of classes inside the package that depend upon classes outside the package. Measured using [aopmetric].	1	
Instability[6]	The ratio of efferent coupling to total coupling. (I=*Efferent*/(*Afferent*+*Efferent*)). Range 0-1 where I=0 is a maximal stable package. Measured using [aopmetric].	1	
Abstractness[6]	The ratio of the number of abstract classes to the total number of classes in package. This metric range is [0, 1]. 0 means concrete package and 1 means completely abstract package. Measured using [aopmetric].	1	
Normalized distance from main sequence[6]	This is the normalized perpendicular distance of the package from the idealized line *Abstractness* + *Instability* = 1. This metric is an indicator of the package's balance between abstractness and stability. Range is [0,1], a value of zero indicates perfect package design. Measured using [aopmetric].	1	
Software process efficiency	Number of retransmissions after acceptance test	1	

indicator[9]			
Functionality[10]	Subjective measurement	1	
Confidence[10]	Subjective measurement	1	
Switching between roles[12]	Count of how many times subjects switch between driver and navigator role	1	
Design quality[13]	Non-comment line of code per method	1	
Knowledge transfer[13]	Subjective measurement	1	
Satisfaction[14]	Subjective measurement	1	
Cohesion[15]	Number of class members generated	1	

1 [arisholm_etal_07]
2 [baheti_etal_02]
3 [canfora_etal_05]
4 [canfora_etal_06]
5 [heiberg_etal_03]
6 [madeyski_06]
7 [muller_04]
8 [muller_05]
9 [nawrocki_woj_01]
10 [nosek_98]
11 [phongl_boehm_06]
12 [rostaher_her_02]
13 [vanhanen_lass_05]
14 [williams_etal_00]
15 [xu_rajlich_06]

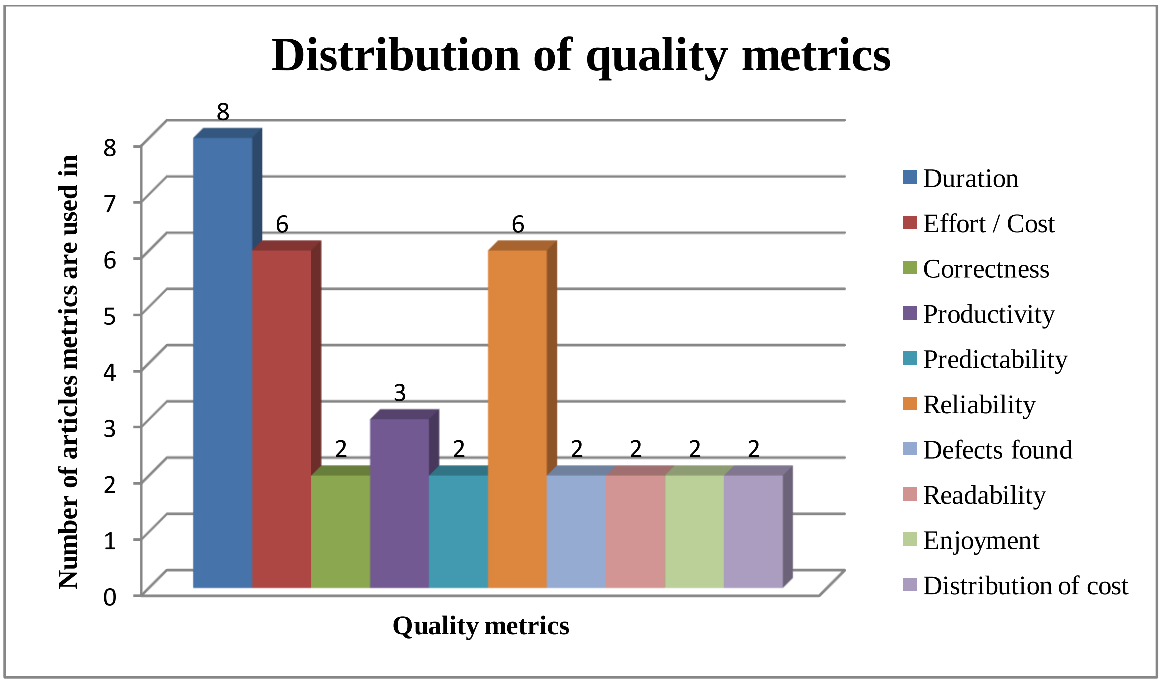


Figure 6: Most common used metrics (metrics that are used in 2 or more studies)

As one can see in Table 4 and on Figure 6, the most frequent quality metrics in the set of articles analyzed in this study is time used on task, which is used in 8 of the 15 studies. Effort/Cost and reliability is the second most used metrics, which are used in 6 of the studies each. The next metric is productivity which is used in 3 of the studies, while correctness, predictability, defects found, readability, enjoyment and distribution of cost all are mentioned in 2 of the articles. Other metrics are only used once, and there for is not considered to be a unifying metric among the articles. In Chapter 5.3 metrics that are used more than once are described and analyzed in detail.

5.3 Detailed analysis of frequently used quality metrics

Following section will elaborate in the most frequently used quality metrics, metrics definition, use in different articles, as well as address the main research questions within the metrics that are used in more than two articles.

> ***RQ1:*** *How is quality defined in a set of articles describing the effect of pair programming technique?*
> ***RQ2:*** *How is quality measured in this set of articles?*

5.3.1 Duration

The most frequent quality metric used in the set of selected articles is "duration", or time used to complete a task. Duration is explained as two things in the set of selected articles. Either as the time taken to complete all tasks considered, or as the total time taken to complete tasks that had been assessed as having passed a certain quality standard. This can be argued whether it is fair to compare these two definitions to each other. Argument for why it isn't fair to compare these two is i.e. a study shows that subjects use 20 min on a task, but the correctness of the task is poor (say 40 % of the solution is wrong), while in another study subjects use 40 min on a task but are not allowed to consider the task finished until they have reached a certain amount of quality, there for the more use of time. This would be the main argument to separate the duration metric into two sub-metrics, "Duration without quality assessment" and "duration with quality assessment". Since the outcomes of the studies are not of great importance to the main research question, this metric will be treated as one metric.

Out of the eight studies that used duration as a quality, almost all of the studies had named the metric differently. Two of the studies called it effort [canfore_etal_05, canfora_etal_06], [phongl_boehm_06] called it "Total Development Cost", and Williams [williams_etal_00] called it "Productivity". Other names such as relative time spent, time used, total development time were also used. Even though this metric turned up to have different names the definition of what was measured was "time used to complete the task(s)".

RQ1: Duration = Time used to complete a task

RQ2: Time used to complete all tasks given with no regard to correctness
Time used to complete all tasks given with regard to correctness

5.3.2 Effort / Cost

The second most used quality metric is somewhat not used correctly. Effort is the same as defined (effort = x amount of time used per subject) but cost can vary a bit. Some programmers can cost more than others so if a team of well paid programmer uses 20 minutes to complete a task, vs. a pair of poorly paid programmers that also use 20 minutes, the cost will vary. Since the cost is not mentioned as a varied metric, and no differences in cost is mentioned from subject to subject it is treated equally.
Effort/cost metric is defined as "time (effort/cost) used per subject" which will state that pair programmers have their time doubled to compute their effort/cost use. For example if a subject uses 20min on a task, then their effort/cost is 20. If two subjects form a pair, and use 20 min on a task their cost/effort equals 40. The reason why effort and cost are united as the same metric is due to its use in

the articles. All articles that mention cost does not mention how they define cost, or how valuable a programmer is. Due to this, these two metrics can be considered as one.

Other names used on this metric is Productivity [vanhanen_lass_05], Total time cost [xu_rajlich_06]. Two articles named this metric as effort, and two named it as cost.

RQ1: Time (effort) used per subject.
Time (cost) used per subject.

RQ2: Effort = Time used per subject to complete the task(s) given.
Cost = Time used per subject to complete the task(s) given multiplied with cost per hour (which is 1, that is why it becomes the same as effort)

Example:
Solo programmer uses 15 hours to complete a task, where pair programmers use 9 hours. Since there are two subjects in a pair, the total cost/effort for the pairs is doubled. In this example I am assuming that the cost for a programmer is 1000 NOK.

	Solo programmer	**Pair programmers**
Time used	15 h	9 h
Effort	15 h	18 h
Cost	15 000 NOK	18 000 NOK

5.3.3 Reliability

Along with Effort/Cost metric this is the second most used. Reliability is used in 6 of the 15 studies, and is defined as “test cases passed”. The more test cases are passed thus more reliable the solution is. According to [rel_wiki] IEEE defines reliability as “. . . the ability of a system or component to perform its required functions under stated conditions for a specified period of time." So the use of the “test case passed” definition is not far away from how the IEEE defines the metric. The logic behind this is that the more test cases are passed thus more fault-tolerant the solution is, which leads to the ability of a system to perform its required functions for a specified period of time. The qualities of test cases used to evaluate this metric are not taken into consideration.

Other names used for this metric are, “Quality of the solution” [heiberg_etal_03], Un-passed test cases [phongl_boehm_06], “Development Quality” [williams_etal_00], robustness [xu_rajlich_06]. [heiberg_etal_03] used “Quality of the solution” as the only quality metric in his study. Also worth mentioning is the “un-passed test cases” used in [phongl_boehm_06], which is kind of an inverted test cases passed. In the end the authors measure the same thing and that is why it is considered to be an equal to test cases passed.

RQ1: Test cases passed
RQ2: Reliability = Test cases passed / Total test cases

More cases passed indicate a more reliable solution.

5.3.4 Productivity / Efficiency

Productivity is defined as lines of code per hour, and it is the fourth most used metric in this selection of experiments. Two of the studies [williams_etal_00] and [vanhannen_lass_05] use this term differently. According to [williams_etal_00] productivity is the same as duration, time used to complete a task, and in [vanhannen_lass_05] it is defined as "x amount of time used per subject", in other words same as effort/cost. Productivity is defined as the quality of being productive or having the power to produce [dict_prod_06]. The three experiments that mention productivity, define it as "line of code per hour".

Of the 3 articles that mention productivity, only Baheti et.al [baheti_etal_02] call it productivity. Nawrocki [nawrocki_woj_01] and Xu and Rajlich [xu_rajlich_06], respectively, call it program efficiency and efficiency.

RQ1: Line of Code per hour (LOC / hour)
RQ2: Counting number of lines of code produced per hour

The authors of selected articles fail to mention how they actually perform the counting of lines. Whether they use some sort of automated counting, or just count lines of code, and divide it by how many hours were used to finish the assignment.

5.3.5 Correctness

Correctness is mentioned as a metric in two articles [aristholm_etal_07] and [phongl_boehm_06] where it has been defined as "A binary functional correctness score with value "1" if all change tasks were implemented correctly" [arisholm_etal_07], and "Overall project score measured from all previous measurements combined" [phongl_boehm_06]. [aristholm_etal_07] definition of this metric is quite accurate, since the common definition of correctness is a software product that should execute all tasks, which are defined in the requirements and specifications. Phongl [phongl_boehm_06] describes it somewhat differently, and can lead to several interpretations of the definition.

RQ1: A binary functional correctness score with value "1" if all change tasks were implemented correctly / Overall project score
RQ2: Somewhat subjectively measured since the teacher assistants/authors had to decide whether a requirement was fulfilled

5.3.6 Predictability

Predictability means the quality of being predictable. To predict something is a statement foretelling the possible outcome(s) of an event, process, or experiment. A prediction is based on observations, experience, and scientific reasoning [gloss_01]. This metric is used by one author but in two different experiments, and the requirements for accepting a metric was that it was used in more than two articles, which was the case here. [canfora_etal_05] and [canfora_etal_06] defines this metric as "standard deviation derived from time used". In both cases Canfora has calculated the standard deviation from time used. This applies both for solo programmers and for pairs. The reason why this metric was used is to see whether one of the programming techniques is more predictable than the other. When one calculates the standard deviation one tries to see whether there is a consistency in time used by each subject, or whether there are few or many outliers that deviate from the mean.

Example:
Ten programmers use 2h each to finish a task. Here the standard deviation would be equal to zero, because all of the subjects are equal to the mean. If one of the 10 programmers finishes the task in three hours and the other in two hours, then we would have an outlier, and the standard deviation would not be zero anymore. If many data points are close to the mean, the standard deviation is small; if many data points are far from the mean, then the standard deviation is large. If all data values are equal, then the standard deviation is zero. Thus smaller the standard deviation is thus more predictable the outcome is.

RQ1: Ability to predict how much time a programmer using programming technique X uses to finish a task
RQ2: Standard deviation derived from time used

It is no surprise that in both articles [canfora_etal_05] and [canfora_etal_06] the same name for this metric is used, predictability.

5.3.7 Defects

Defect can be defined as "a part, product, or service that does not conform to specification or customer expectations" [mayo_08]. Two of the articles mention this metric [vanhanen_lass_05] and [phongl_boehm_06]. Both these two articles have defined a defect as a non conformation to the specification. The measurements in both articles are somewhat subjective. Phongl and Boehm [phongl_boehm_06] uses teacher assistants to count defects found by them after each delivery phase, and Vanhanen [vanhanen_lass_05] mentions that the defects were counted pre- and post-delivery. In both cases the teacher assistants found and counted the defects, and neither of the two authors mentions what they characterize as a defect. Is it specification defects, algorithm defects, expression defects or is it all? There it is considered to be a subjective measurement.

RQ1: General defects: Non-conformation to the specification. Not explained which type of defects are measured (counted).
RQ2: Defects counted by teacher assistants

Phongl and Boehm [phongl_boehm_06] mentions this metric as defects found, while Vanhanen [vanhanen_lass_05] uses the term defect rate.

5.3.8 Readability

Readability can be defined as "the quality of written language that makes it easy to read and understand" [dict_read_06]. Two articles mention this metric as a part of the quality term, [xu_rajlich_06] and [nosek_98]. According to Xu and Rajlich [xu_rajlich_06] readability is the same as "more meaningful variable names" which is measured by the authors. They fail to mention what are the characteristics of a good or meaningful variable name, and therefore this can without doubt be considered a subjective measurement. To evaluate readability variable, the subjects were asked to properly comment on each of the processes within the script they were programming [nosek_98]. This is how Nosek [nosek_98] interprets readability in his study. In other words commenting properly on each process and one has achieved good readability. This is also a subjective measurement since the author has not mentioned what a good commenting consist of, and what kind of guidelines he has followed to conclude what is a good comment and what is not.

RQ1: Nosek [nosek_98] defined this metric as properly commented process, while Xu and Rajlich [xu_rajlich_06] defines it as "meaningful variable name".

RQ2: Even though the authors do not share the same definition of readability, both their measurements were subjective.

5.3.9 Enjoyment

Enjoyment is defined as "the pleasure felt when having a good time" [dict_enjoy_06]. Two of the articles mention this metric, [vanhanen_lass_05] and [nosek_98]. Both of the authors use it as a prerequisite for developing a product with better quality. I.e. if a programmer enjoys to work in pairs more than alone, this can lead to better quality overall. Things like efficiency, productivity etc can improve from subject enjoying his/her work. One becomes more thorough and passionate in ones work.

RQ1: Enjoyment of working with a specific programming technique.

RQ2: Subjective measurement, interviews/questionnaires done by authors after the assignments.

5.3.10 Distribution of cost

Two of the articles use this metric to find out how much time a programmer spends on developing, testing, and refactoring. Rostaher [rostaher_her_02] define it as "relative time spent for specific programming activity to find out how much time a pair or individual uses on testing, adding new functionality and refactoring, with developer experience as a factor." Phongl and Boehm define it as "Distribution of Cost shows us how the two groups distributed their development, how much time was used in each phase of the project. For example pairs used more time in production, but almost no effort on rework and review (since this was done by the co-driver)" [phongl_boehm_06].

RQ1: Time used in a specific part of the development phase

RQ2: Dist. Cost = Time used in a specific part of the development phase / Total time used

Phongl and Boehm [phongl_boehm_06] calls this metric "distribution of cost", while Rostaher call it "relative time spent for specific programming activity" [rostaher_her_02].

5.4 Classification of quality metrics

5.4.1 Process and product quality

Since the authors of the experiments that were analyzed did not have a unified view on quality, I have decided to categorize the quality metrics found further. In this section all of the quality metrics are used to map more thoroughly what kind of quality metrics the authors used in their studies.

I have come to the decision that 3 categories are needed to place all of the quality metrics,

- Process quality metrics
- Process quality enhancement
- Product quality metrics

Process quality metrics deals with the quality metrics that affects development process. A process can be defined as "a series of operations performed in the making or treatment of a product."
The process quality enhancement metrics does not affect the process directly, but they can tend to improve the process quality indirectly. I.e. if the communication within the team is at a high level, and there are rarely any misunderstandings, the productivity of the team is likely to improve. It is also noticeable that all of these metrics are measured subjectively. Since they affect the development process indirectly I have decided to place these metrics in an own group. Product quality metrics affect the product, which is no surprise. A product can be defined as "an artifact that has been created by someone or some process". Table 5 presents the found quality metrics classified in their respective fields.

Table 5: Classification of quality metrics

Process quality metrics	**Process quality enhancement**	**Product quality metrics**
Effort[1,14] / Cost[7,8]	Knowledge transfer[13]	Cohesion[15]
Productivity[2]	Satisfaction[14]	Design quality[13]
Predictability[3,4]	Confidence[10]	Functionality[10]
Duration[1]	Communication within the team[2]	Afferent coupling[6]
Distribution of cost[11]	Cooperation within the team[2]	Efferent coupling[6]
Software process efficiency indicator[9]	Enjoyment[10,13]	Instability[6]
	"Quality"[2]	Abstractness[6]
		Normalized distance from main sequence[6]
		Program length[15]
		Readability[10]
		Defects found[11]
		Reliability[7,8]
		Correctness[1]

1 [arisholm_etal_07]	6 [madeyski_06]	11 [phongl_boehm_06]
2 [baheti_etal_02]	7 [muller_04]	12 [rostaher_her_02]
3 [canfora_etal_05]	8 [muller_05]	13 [vanhanen_lass_05]
4 [canfora_etal_06]	9 [nawrocki_woj_01]	14 [williams_etal_00]
5 [heiberg_etal_03]	10 [nosek_98]	15 [xu_rajlich_06]

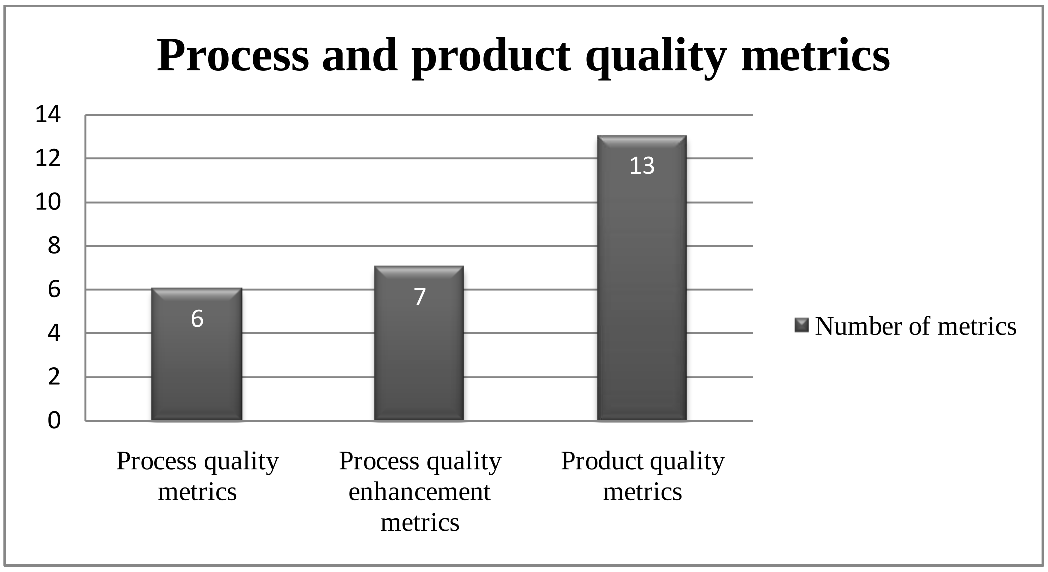


Figure 7: Classification of quality metrics

Out of the 26 quality metrics found in the studies (see Figure 7),

- 6 are classified as process quality metrics
- 7 are classified as process quality enhancement metrics
- 13 are classified as product quality metrics

5.4.2 Subjective vs. Objective measurements

Out of the 26 quality metrics, I have classified 11 of those as subjective measurements (see Table 6 and Figure 8). For a metric to be regarded subjective there has to be human involvement in the measurement process, and if we repeat the measurement of the same object(s) several times, we would not get exactly the same measured value every time [inf5180]. Two of the metrics which are regarded subjective could in some sense be regarded objective. These are duration and defects found. The reason for why I have chosen to classify these as subjective is because there is a lack of explanation by the authors who use these metrics of how it is actually measured. Duration is most likely measured “by hand”, and I doubt that the authors used some automated way for measuring this. “By hand” I mean that an approximate measurement is done, and not an automated. “Defects found”, here the authors counted the defects, and whether the defect count would be different if another person counted them is uncertain but most likely, since the authors did not specify what kind of defects are counted and what kind are not. 15 of the metrics are classified as objective metrics on the ground that the metrics are automated or the measuring process is (almost) perfectly reliable (see Table 6 and Figure 8).

Table 6: Overview of subjective and objective measurements

Subjective	**Objective**
Knowledge transfer[13]	Cohesion[15]
Defects found[11]	Distribution of cost[11]
Functionality[10]	Productivity[2]
Satisfaction[14]	Duration[1]
Confidence[10]	Effort[1,14] / Cost[7,8]
Communication within the team[2]	Correctness[1]
Cooperation within the team[2]	Predictability[3,4]
Enjoyment[10,13]	Afferent coupling[6]
Readability[10]	Efferent coupling[6]
"Quality"[2]	Instability[6]
Software process efficiency indicator[9]	Abstractness[6]
	Normalized distance from main sequence[6]
	Program length[15]
	Design quality[13]
	Reliability[7,8]

1 [arisholm_etal_07]	6 [madeyski_06]	11 [phongl_boehm_06]
2 [baheti_etal_02]	7 [muller_04]	12 [rostaher_her_02]
3 [canfora_etal_05]	8 [muller_05]	13 [vanhanen_lass_05]
4 [canfora_etal_06]	9 [nawrocki_woj_01]	14 [williams_etal_00]
5 [heiberg_etal_03]	10 [nosek_98]	15 [xu_rajlich_06]

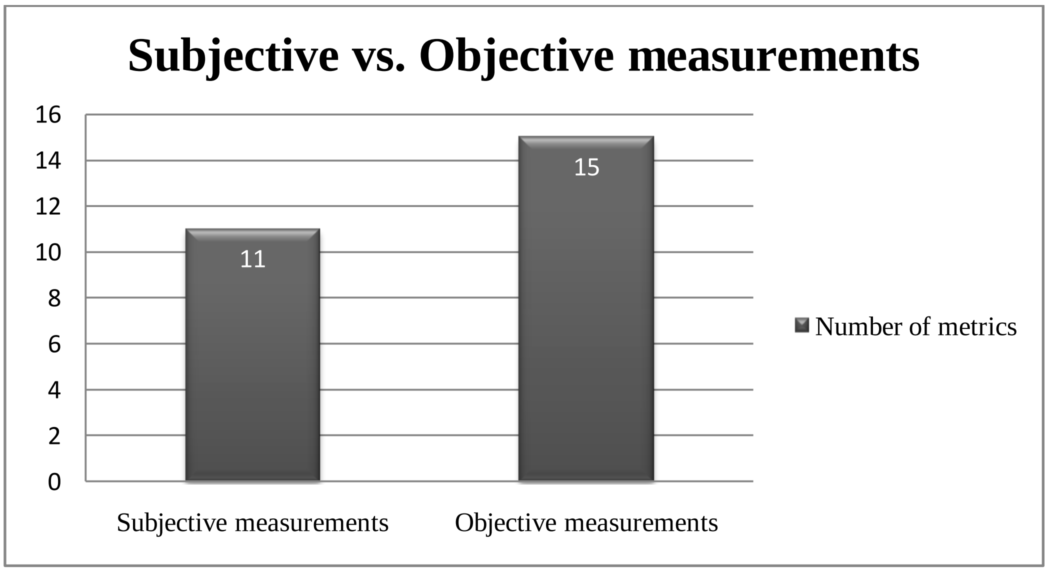


Figure 8: Overview of subjective and objective measurements

An overview is also presented (see Figure 9) of the metrics mentioned in Chapter 5.3 where I discuss the metrics found in more than one article. 10 metrics were used in more than one article, out of these 4 can be classified as product quality metrics, 5 belong to the classification process quality metrics, and 1 is classified as process quality enhancement metric. Among these 10 metrics, 5 are measured subjectively and 5 are measured objectively.

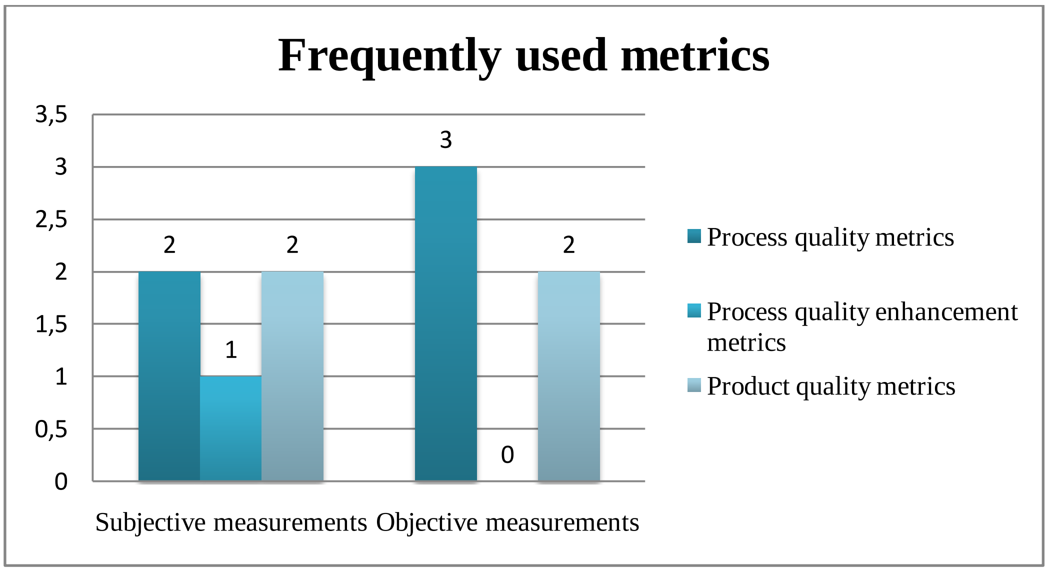


Figure 9: 10 most used metrics classified into subjective/objective measurements and process/product quality

6 Discussion

6.1 Definition and measurement of quality

The research questions of this thesis requests an overview of how the set of articles selected define and measure quality.

> ***RQ1:*** *How is quality defined in a set of articles describing the effect of pair programming technique?*
> ***RQ2:*** *How is quality measured in this set of articles?*

6.2 What authors call quality

In the set of articles that have been analyzed all of the authors make use of the term quality in some way, yet none of them have taken the time to explain what kind of quality they mean, what are they trying to measure, prove etc. I.e. Williams [williams_etal_00] states "…two programmers working side by side at one computer on the same design, algorithm, code or test – does indeed improve software quality…". In this case what does the author mean by "software quality"? It is not so obvious. Further into the article one can see that the author measure defect rate, effort, satisfaction and time used on an assignment or productivity as they call it. With some reasoning I will conclude that Williams, in [williams_etal_00], defines quality in two segments. The process quality which include the productivity (time used in this case) and effort. Satisfaction is a subjective measurement, which can affect the process quality, and there for considered to be a part of the process quality enhancement metrics. Defect rate is part of product quality which measures the robustness of the system, measured by test cases passed. This is just one the examples where the authors lacks an explanation of the term quality. This leads to assumptions among the readers and the interpretation can vary, which again leads to more confusion and difficulty of unifying a common term for quality within the software engineering field.

Also none of the authors have given any reason why they include some types of quality metrics or why they exclude other types of quality metrics. All the studies are about the efficiency of pair programming and yet out of 26 different quality metrics found among these articles, only 10 of these are used in 2 articles or more. That leaves 16 quality metrics which are used in only one article. These authors are regarded as the leading researchers in their respective fields, but as it seems even they cannot agree on how to use the term quality in a common and unifying manner. It is then no wonder that an "ordinary" researcher, a company or an individual cannot agree on a unanimous explanation or description on the term quality.

One can read in Chapter 5.2 and see Figure 6 what is the most common use of the term quality among the authors. Chapter 5.3 elaborates more on the quality metrics that are used in more than 2 articles.

6.3 Affiliation of authors

It is interesting to see whether some of the authors are biased towards one programming technique, in this case, pair programming, and the answer is yes as everyone anticipated. What is more surprising is how many of the authors biased towards pair programming. In Figure 10 “Overview of authors’ affiliation” we can see that approximately half of the authors are biased.

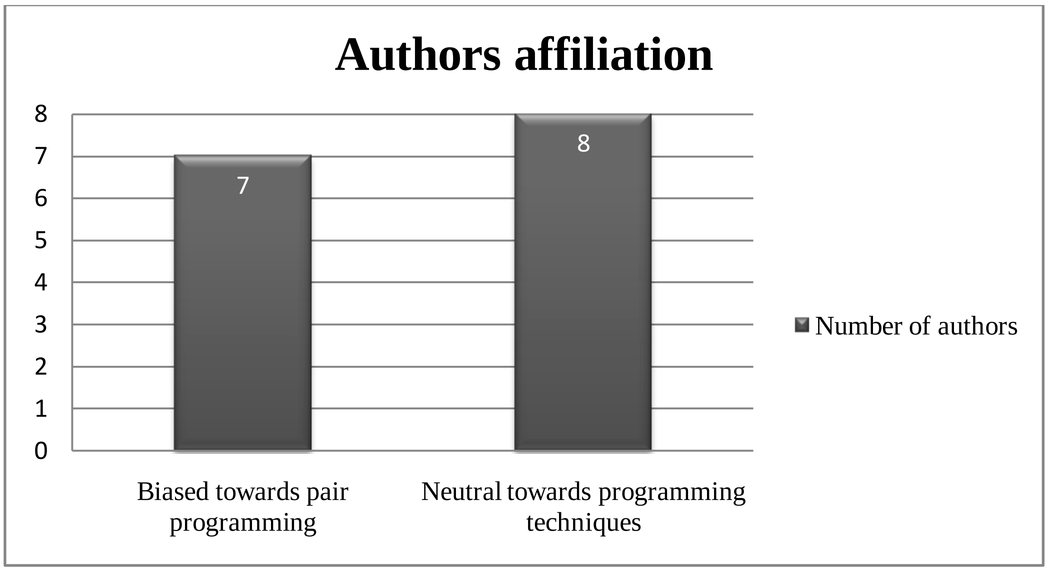


Figure 10: Overview of authors’ affiliation

Authors that are biased towards pair programming:

Baheti [baheti_etal_02], Canfora [canfora_etal_05], Canfora [canfora_etal_06], Heiberg [heiberg_etal_03], Nosek [nosek_98], Rostaher [rostaher_her_02], Williams [williams_etal_00]

Authors that are neutral towards programming techniques:

Arisholm [arisholm_etal_07], Nawrocki [nawrocki_woj_01], Madeyski [madeyski_06], Muller [muller_04], Muller [muller_05], Phongl and Boehm [phongl_boehm_06], Vanhanen [vanhanen_lass_05], Xu and Rajlich [xu_rajlich_06]

Now that we have an overview of which authors are biased and which are neutral, it is interesting to see what kind of metrics they used in their studies, and investigate whether different metrics are used by the authors which are biased and those that are not. In addition to which metrics are used, I have also investigated whether those metrics are subjective or objective measurement. This way we can get a picture of whether authors that are neutral or biased towards pair programming use different metrics as evidence to underlie their claim.

As we can see from Table 7 and Figure 11 the authors that are neutral towards the programming techniques use mostly objective metrics to prove their claim. The total number of metrics used by these authors is 19, and 26% of them are subjective metrics. The authors that are biased towards pair programming use in total 13 metrics as evidence to claim their point. Out of these 13 metrics, 62% are subjective. This is shown in detail in Table 8 and an overview is found in Figure 11.

Table 7: Metrics used by authors neutral towards programming technique

Subjective	**Objective**
Knowledge transfer[13]	Cohesion[15]
Defects found[11, 13]	Distribution of cost[11]
Enjoyment[13]	Productivity[2, 9, 15]
Readability[15]	Duration
Software process efficiency indicator[9]	Effort[1] / Cost[7,8, 13, 15]
	Correctness[1, 11]
	Afferent coupling[6]
	Efferent coupling[6]
	Instability[6]
	Abstractness[6]
	Normalized distance from main sequence[6]
	Program length[15]
	Design quality[13]
	Reliability[7,8, 11, 15]

Table 8: Metrics used by authors bias towards pair programming

Subjective	**Objective**
Functionality[10]	Duration
Satisfaction[14]	Effort[14]
Confidence[10]	Distribution of cost[12]
Communication within the team[2]	Predictability[3,4]
Cooperation within the team[2]	Reliability[5,14]
Enjoyment[10]	
Readability[10]	
"Quality"[2]	

1 [arisholm_etal_07]	6 [madeyski_06]	11 [phongl_boehm_06]
2 [baheti_etal_02]	7 [muller_04]	12 [rostaher_her_02]
3 [canfora_etal_05]	8 [muller_05]	13 [vanhanen_lass_05]
4 [canfora_etal_06]	9 [nawrocki_woj_01]	14 [williams_etal_00]
5 [heiberg_etal_03]	10 [nosek_98]	15 [xu_rajlich_06]

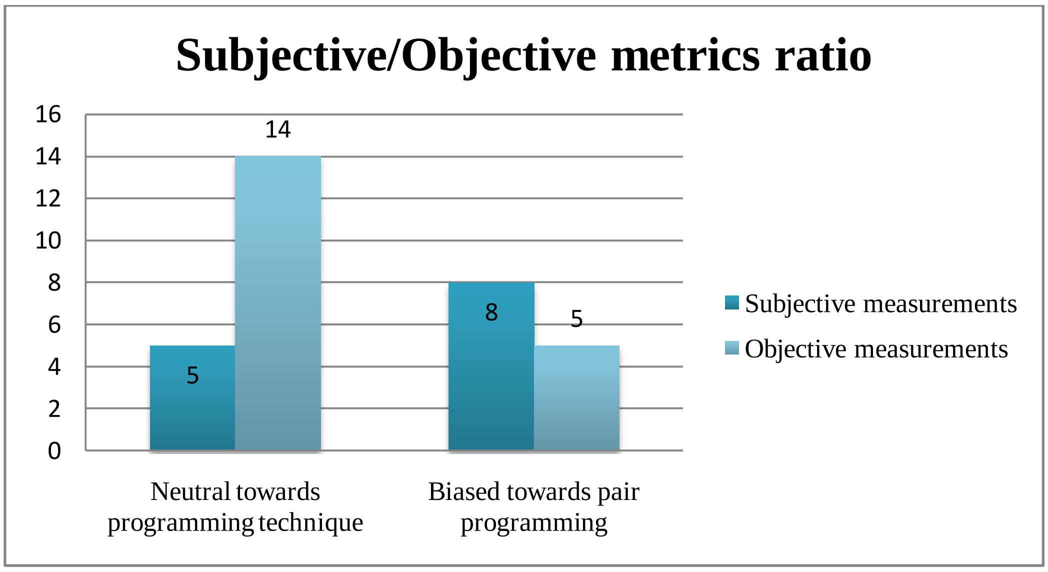


Figure 11: Overview of biased/unbiased authors and their measurements

Since the authors did not use same amount of metrics it is only fair to display these in percentage. What one should have in mind is that, even though the authors that are neutral to pair programming used more metrics to strengthen their claim, this is done only at a small scale, so irregularities can occur.

Authors neutral towards pair programming:

- 26 % subjective metrics
- 74 % objective metrics

Authors bias towards pair programming:

- 62 % subjective metrics
- 38 % objective metrics

6.4 Term confusion regarding metrics

There is no surprise that there is going to be a confusion regarding terms when it comes to quality. What one person means when he states that a product is of high quality is not necessarily the same as what another person means with the same sentence. Comparing the terms used in this investigation I have discovered some interesting things. Not only did the authors use different metrics when trying to prove that something is of better quality, they also used different terms for the same metrics. No wonder there is still so much confusion in the software engineering field. Table 9 presents the reader with an overview of the misuse of different terms. The most misused term seems to be “productivity”. Authors uses this term as a term for measuring time used to complete a task, cost per subject, and lines of code written per hour. So when an author states that his way of doing a task is more productive than something else, one should definitely try to understand what the author means with productivity. Another example would be, e.g., different terms for reliability. Some authors call this metric “quality of the solution” or “Development quality” etc. More examples of different use of the same term, or different terms of the same measurement one can read in Table 9.

Table 9: Metrics and term confusion among authors

Quality metric	**Definition**	**Other names for the same metric**
Duration[1]	Time used to complete the task	Effort[3,4], Total Development Time[9], time used[10], Total Development Cost[11], Relative time spent[12], Productivity[14]
Effort[1,14] / Cost[7,8]	Effort/Cost in x amount of time per subject	Productivity[13], Total time cost[15]
Correctness[1]	Binary functional correctness score	Project score[11]
Productivity[2]	Line of code per hour	Program efficiency[9], Efficiency[15]
Reliability[7,8]	Test cases passed	"Quality of the solution"[5], Un-passed test cases[11], "Development Quality"[14], Robustness[15]
Readability[10]	Subjective measurement	More meaningful variable names[15]
Distribution of cost[11]	Measured how much time was used in different phases of the development cycle	Relative time spent for specific programming activity[12]
"Quality"[2]	Average grade obtain by the group	
Software process efficiency indicator[9]	Number of retransmissions after acceptance test	
Design quality[13]	Non-comment line of code per method	

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 15 articles that reported on the effectiveness of pair programming. The authors of [dybå_etal_07] searched the ACM Digital Library, Compendex, IEEE Xplore, and ISI Web of Science with the following basic search string: "pair programming" OR "collaborative programming." This search strategy opens the risk of a validity threat consisting of the loss of good articles that do not refer to themselves as one of those search strings, and yet fulfills the criteria of being one. In addition, the authors hand-searched all volumes of the following thematic conferences proceedings for research papers: XP, XP/Agile Universe, and Agile Development Conference.

7.2 Data extraction

During the analysis I extracted data from 15 articles. The data provided answers to various questions in the thesis. Considering the lack of guidelines and standards on how to conduct an experiment and focus on quality in empirical software engineering, extracting the answers and concluding the findings, and furthermore classifying the metrics found showed to be sometimes difficult. It was not always obvious what the answers and explanations were. Therefore the answers had to be interpreted. Due to the fact that the analysis had to be carried out be one person only the data may include some subjective interpretations of the data. 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. Since this was not something that could be reviewed by several reviewers, the finding can contain some misclassifications. To minimize the probability of extracting different data from the articles 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

This thesis studied the concept of the term quality used in a set of selected experiments on the effectiveness of pair programming. I presented an overview that characterizes what authors and researchers call quality in empirical software engineering. On the basis of this overview, other authors and researchers may decide further research for improving and narrow the use of term quality.

The objective of the investigation was to get an overview of how quality is defined, described and measured in experiments which study the effect of pair programming.

In order to address these issues, I analyzed a set of articles that study the effect of pair programming. The set consist of 15 articles, which are selected from a larger set (214 articles). In Chapter 4.2 review protocol, it is explained how these 15 articles were selected. The data collected during the analysis of these articles, was used to answer the research questions.

8.2 Findings

Population:

- The number of subjects varied from 12 to 295, with a median of 24.
  11 of the studies used students as subjects, while 4 used professionals.

Outcome of the studies:

- Outcomes of the studies varied and were inconsistent with each other.

Bias among the authors:

- There were a lot of bias among the authors of the studies, and it seemed that many authors favored a programming technique before they conducted the experiment. Some of the authors are advocates for the pair programming cause, and this comes out clearly in their reporting, where they try to magnify their results to present their cause as positive as possible.

Term confusion:

- There was a lot term confusion regarding quality metrics among the authors of the articles. Many used different terms for measuring the same metric, and same metric terms for measuring different metrics. More detailed overview of this can be read in Chapter 6.1.3.

Quality metrics:

- There were used 26 different quality metrics in the 15 articles.

Most frequently used quality metrics:

- *Duration* was the most frequently used metric, which was used in 8 of the 15 articles.
- *Effort and cost* together with *reliability* was used in 6 articles, and 7 other metrics were used in two articles or more. This leaves 16 unique quality metrics which were used in only one article.

Process and product quality metrics:
The quality metrics were categorized into three groups:
- Process quality metrics (6 metrics)
- Process quality enhancement metrics (7 metrics)
- Product quality metrics (13 metrics)

Subjective and objective measurements:
- The 26 metrics were further categorized into either subjective or objective metrics. 11 metrics fitted the category subjective measurements, while 15 fitted objective measurements.

Affiliations of authors:
- Out of the 15 articles, 7 of the articles' authors were biased towards pair programming, while 8 of them remained neutral towards programming techniques.

Author's affiliation and use of measurements:
- The authors who were unbiased towards a programming technique used in total 19 different metrics to describe quality, out of these 5 of them were subjective measurements, and 14 were objective. While the authors who were biased towards pair programming used 13 different metrics, and out of these, 8 were subjective measurements and 5 were objective.

8.3 Discussion

The findings presented in previous section shows that there are great differences and variances in software engineering studies regarding pair programming. Not only is there a variance, as expected in outcomes but there is a great variance also in number and type of subjects used, quality definitions, metrics used, affiliations of the authors, and last but not least claims based on different measurements.

The variance in type and number of subjects is expected from study to study, but not as much as shown in this thesis. The numbers vary from 12 to 295 with a median of 24. I will use the median number to show how many subjects it produces. Usually we want to have equally distributed groups, which in this case mean we have to have 8 pairs and 8 individuals. In best case if all of the subjects deliver what is expected of them, and actually finish all the tasks in an experiment, we still have a minimal amount of subjects to be certain whether the results produced are affected by the small number of subjects. With so few subjects it is hard to conclude whether the results are a product of too few participants, or that these are actual results.

Surprisingly many authors were biased in the articles used in this thesis. It seemed that the only agenda they had was to persuade readers to share their opinion and view on a matter, instead of presenting the issue without shining the light favorably on one side of the case. In other words present both cons and pros and let the reader take a standing point in the issue.

None of the authors explained what their criteria were for including or excluding the metrics they used in the studies.

The authors claiming that a certain programming technique brings much more quality to the end product do not state explicitly or vaguely what they mean with the phrase "better quality". One has to dig through the whole article, draw conclusions regarding use of quality metrics and analyze thoroughly to find out what the author(s) are talking about when they claim something to be of a "better quality". It is no wondered that researchers and professionals in the industry are confused when it comes to use of the term quality.

To make the use of quality metrics even more difficult to understand the authors tend to use quality metrics of different types. With different types I mean process quality, process quality enhancements and product quality metrics. For instance one author use only process quality metrics to claim that pair programming is of "better quality" while another use, for example, product quality to claim the opposite. This leads to even more confusion regarding this issue. In addition to that there is a difference among the authors on use of subjective and objective measurements in their research. It is shown that authors, who are advocates of pair programming, use more subjective measurements to claim that pair programming is "better" than solo programming, than authors who are neutral to the subject.

I have learned that throughout the articles regarding pair programming efficiency the authors inconsistently use quality definitions and measurements and later use these to claim their results to be "valid". This inconsistency seems to take overhand and if not viewed upon with more critical eyes it could lead to a step back in maturing the software engineering community and research. For instance authors measures one and the same thing but they call it differently in different articles, or they use the same name for a metric but the metric does not measure the same thing. If authors do not explicitly state what they want to measure, how they will measure it, elaborate on criteria for including or excluding some quality measurements when it is obvious to use or not to use certain metrics and so forth, the software engineering community can experience a negative effect and also produce a lot of articles that are of poor quality. As the findings from this thesis have shown it seems that there is a long way ahead of us in accomplishing an agreement on what standards to use when i.e. claiming that a programming technique holds better product quality than another. When leading researchers fail to define, measure, and use the quality metrics correctly, it should not come as a surprise that students and other researchers mix up and treat these issues inconsistently. It all seems simple, but in reality it is extremely difficult to get a group of people, in this case the researchers to agree on a set of standards.

These findings should act as a wakeup call to the software engineering community, and to the research field of pair programming. Everything points towards that the software engineering research community needs a lot more maturing, and a lot of more experiments to grow towards a reliable source of knowledge. Most of the studies done up to today are premature in many aspects. From not agreeing on common and universal standards, lack of knowledge on how to conduct unbiased experiments, to blindly accepting findings and claims done by authors who have a foot planted inside the software engineering research industry. To become a mature research environment, more studies are needed on this field, not to mention more unbiased studies are needed. Also fellow researchers have to be more critical towards each other's work. Not to undermine a fellow researchers work, but to increase the quality of the study by reviewing it with critical eyes.

I hope that the result of this thesis will be useful to researchers who are looking to prove that a programming technique or something else in the matter is of a better quality than something else. This thesis could also be interesting in a way to the industry because it shines a light on how quality is defined, measured, and not at least used in a set of articles. Perhaps some companies have relied on some of the papers used in this thesis to come to a conclusion on whether to use pair programming or solo programming actively as a programming technique. With the results enquired it is suggested that authors who use the term "better quality", without explicitly explaining what it means and how it is measured to achieve “better quality”, in future tries to address this issue. To the readers it is suggested to read with more critical eyes, whoever writes the articles.

8.4 Future work

As the findings have shown there is a great difference of defining and measuring quality in this set of articles, presented in this thesis. Hence, there is a need of a standardized way of conducting experiments when quality is in focus. A tailored definition should be proposed and tried out in order to initiate the improvements. Such a definition will be of great importance in future research conducted by other researchers.

In the future it would be of great interest to look into more articles not only regarding pair programming but also other aspects of the software engineering and see how quality is defined and measured there. Also to validate my findings there is need of expanding this review to contain more articles, more reviewers to eliminate all possibilities of bias and doubt in the results.

Appendix A – Detailed analysis of the selected articles

Empirical Validation of Test-Driven Pair Programming in Game Development

Shaochun Xu, Vaclav Rajlich

IEEE 2006

Article

Abstract
Investigate pair programming, test-driven development and refactoring in game development. Test results show that pairs wrote higher quality code and completed tasks faster.

Introduction
Mention that pair programming has been used in the industry and reported some promising results, refers to Nosek [nosek_98].

Pair programming in game development
"Pair programming has been widely known with its fast development cycle and high quality code [williams_etal_00]." [xu_rajlich_06]
[nosek_98] found that all pairs outperformed the individuals in terms of quality and time spent (industry study), [williams_etal_00] were more confident in their solution. Williams and Upchurch found also out that pairs communicated more efficiently, learned faster and were happier.
[nawrocki_woj_01] reported that pairs spend twice as much total effort than individuals, and Beck mentioned in his findings that pair programming is not suitable for very large projects.
Collaboration is an important issue in game design because it affects the quality of the games and production time [xu_rajlich_06].

TDD
In TDD programmers write test cases for the new functionality before writing the code. "The number of test cases passed vs. the total number of tests is a metric to show the quality of the products." [xu_rajlich_06]

Refactoring
Refactoring means consistently cleaning and improving the code which makes code easier to maintain and extend. Applying refactoring during the game development can enhance game quality, since clear code takes less time to run and compile, as well as making debugging easier.

Case Study
Task is to implement an application which records the scores for bowling games. Note that this kind of application will ONLY record the score. It is not mentioned that any graphics are implemented, which is essential to almost every game.

In order to trace development process and time spent "Microsoft Producer" was used. Both groups were asked to write a high quality program in an efficient way.

Results and discussion
"Please note that the times listed in Table 1 and 2 are the duration that the pairs and individuals spent on the task, therefore the total time cost for a pair is twice as much as the time indicated." – In the table is the time used, while the cost would be twice as high for pairs. The tables mentioned here refer to the articles tables not the tables used earlier in this thesis.
Main characteristics mentioned in the table 1 are: LOC, Number of class members, number of classes, number of classes created in first half of code, numbers of refactoring, numbers of test cases passed, time used, and LOC pr. Hour.
Results from table 1: Pairs wrote more LOC, but had more than twice as much LOC written per hour than individuals, which indicates higher efficiency. Pairs had higher cohesion which is also supported by the findings Beck [beck_03] reported. Here pairs had more than twice as many class members then the individuals. Since Beck did a report on TDD, this can be due to pairs used of TDD technique, rather than an effect of pair programming. Authors claim that pairs' class members are also much more elegant and readable, as well as having more meaningful variable names. Neither of these claims is supported with any evidence. In order to test software's robustness the authors created 12 black-box tests. Almost all the pairs passed 12 of 12 tests (11,8), while individuals had an average of 9.8 / 12.
According to properly use of TDD technique, the developers design the tests prior to coding the software, and the question is whether pairs got a sneak preview of the 12 black box tests prior to writing the code. If so, it could have lead to bias because they knew where the "catches" were. Note that this done in an academic setting, and assuming that the students grades depended on the result, perhaps they used they knowledge of how these tests work to develop a better solution that they would have had.

Limitations:
Since this was a simple game application it is not sure that these results can be reproduced if one starts involving heavy graphics like a "real" game is using. Sample size was also relatively small, and the subjects were novice programmers (which could influence the outcome).

Main Research Questions

Q1	Q2
Program length	LOC counted
Efficiency (development)	LOC written per hour
Cohesion	Class members generated (pairs generated twice as many class members, which implies higher cohesion)
More meaningful variable names	No explanation (Subjective measurement?, even though this is not mentioned)
Robustness	Black box test cases passed
(Time cost)	Total time cost (time used per individual [means time used is double for pairs])

Population selection:
8 undergraduate students from one University and 4 from another. All are classified as novice programmers as well as advanced in their classes. Randomly chose students from the 12 to form pairs.
4 worked alone, and 4 pairs were established. Pairs also applied TDD and refactoring, while the individuals practiced traditional waterfall-like approach.

Outcomes
Study shows that pairs spent much shorter time on in completing the task than individuals. Pairs completed the tasks with "higher quality" since their programs passed more test cases. They also had higher cohesion, and more meaningful variable names. Pairs also wrote much more lines of code per hour than the individuals, showing its efficiency.

Possible motivational biases in estimation
The authors were more or less neutral in the case pair programming vs. solo programming. What can cause a bit bias is that their experiment did not resemble a real game development project. Instead of dealing with a lot of heavy graphics, which is the essential of every game (almost every), they designed a software to keep track of scores and said nothing about whether graphics were involved. The fact that the students were novices in programming rules that probably out. They should used students with some programming background, and designed an experiment where the students programmed an application with modest graphics at least. Another possible bias can be that they included too many variables in the experiment, pair programming, TDD, refactoring, and did not test several combinations of these to see which technique gave the "improved" effect. For instance, solo programming with TDD, pair programming without TDD but with refactoring etc. It is wrong to claim that due to pair programming the pairs had more cohesion in their solution, because one cannot exclude either TDD as a factor which influenced the results. Most probably was the TDD that was the reason for the high cohesion. Because of this small misrepresentation I believe that perhaps the authors introduce a bit bias. They also failed to explain why some quality attributes were left out from the representation (the tables), attributes like cost even though they explained it briefly in the text.

The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design - An Experiment

Lech Madeyski
Springer-Verlag Berlin Heidelberg 2006

Article

Abstract
Test-driven development and pair programming both aim to improve software quality. The objective of this paper is to provide an empirical evidence of the impact of both practices on package dependencies playing a role of package level design quality indicators [madeyski_06].

Introduction
In his previous studies Madeyski found that using TDD had a significant positive impact on two class level software qualities, “Response For a Class” (RFC), and “Coupling Between Object classes” (CBO). He also mentions several other papers on pair programming, TDD and combined studies and gives a summary of their findings. This paper aims to find out the impact of pair programming and TDD on quality of an object oriented design in terms of dependencies between packages, which in turn may have impact on external quality such as fault-proneness or maintainability.
The quality of an object-oriented design is strongly influenced by a system’s package relationship. [madeyski_06] Loosely coupled and highly cohesive packages are qualities of good design.
To investigate the impact of TDD and pair programming on object-oriented design we use Martin’s package level dependency metrics [martin_94, martin_04], that can be used to measure the quality of an object-oriented design in terms of the independencies between the packages of that design.
Martin’s metrics, investigated by the author in this study are: [madeyski_06]

Ca (Afferent Couplings) — the number of classes outside the package that depend upon classes within the package.
– *Ce* (Efferent Couplings) — the number of classes inside the package that depend upon classes outside the package.
– *I* (Instability) — the ratio ($Ce/(Ca+Ce)$) of efferent coupling (*Ce*) to total coupling ($Ce+Ca$). This metric is an indicator of the package’s resilience to change and has the range [0, 1]. $I = 0$ indicates a maximally stable package.
$I = 1$ indicates a maximally instable package.
– *A* (Abstractness) — the ratio of the number of abstract classes to the total number of classes in package. This metric range is [0, 1]. 0 means concrete package and 1 means completely abstract package.
– *Dn* (Normalized Distance from Main Sequence) — this is the normalized perpendicular distance of the package from the idealized line $A + I = 1$. This metric is an indicator of the package’s balance between abstractness and stability. *Dn* metric’s results are within a range of [0, 1]. A value of zero indicates perfect package design.

Quantifiable Hypotheses Formulation

The author rejects the null hypothesis which states that there is no difference in the mean value of X metric (where *X* is *Ca*, *Ce*, *I*, *A* or *Dn*) between the software development projects using any combination of classic (test-last) / TDD (test-first) testing approach and solo / pair programming development method (CS – Classic Solo, TS – TDD Solo, CP – Classic Pair and TP – TDD Pair are used to denote development methods). [madeyski_06]

Design of the Experiment
The assignment of subjects to groups was performed fist by stratifying the subjects with respect to their skill level, measured by grades and then assigning them randomly into either TDD or classical development. The assignment to solo or pairs the author took account on peoples preferences.
Due to 16 teams dropped out (didn't check in their final version and there for were excluded from the analysis) the design of the experiment resulted in unbalanced design.

Validity Evaluation
The author states that no big threats to any kind of validity can be found. Statistical validity is under control due to use of robust statistical techniques, and large sample sizes are used. Measurements and treatment implementation are considered reliable also. One threat to internal validity is a rivalry between the groups, if one group does their outer best to show that an old method is still competitive. However the subjects were informed that their skills were not to be measured in this experiment, only the development technique. The largest threat to external validity is that students were used as subjects. They don't represent the actual population, but since the students are the next generation of software professionals, which is relatively close.

Summary and Conclusion
According to the results it appeared that package level design quality indicators were not significantly affected by development method. Using TDD instead of classic testing approach as well as pair programming instead of solo programming had no significant impact on package dependencies. Earlier results indicated that using TDD instead of classical approach had positive impact on some class level software quality indicators (CBO, RFC) in case of solo and pair programming. [madeyski_06]
Combined results suggest that positive impact of TDD on software quality may be limited to class level. [madeyski_06]

Main research question

Q1	Q2
Afferent Coupling	Measured using aopmetrics
Efferent Coupling	Measured using aopmetrics
Instability	Measured using aopmetrics
Abstractness	Measured using aopmetrics
Normalized distance from Main Sequence	Measured using aopmetrics

The goal of this study was to provide an empirical evidence of the impact of both practices on package dependencies playing a role of package level design quality indicators. Madeyski explained why he only included these types of quality metrics, and excluded others. The quality metrics are explained in the introduction of this article analysis.

”These internal quality attributes may affect external qualities such as fault-proneness (robustness) or maintainability. ” Collected Martin’s metrics using [aopmetric] tool developed and supported by members of e-Informatyka at Wroclaw University of Technology.

Population selection
188 students (academic) participated at the start of the experiment, where 122 finished. These 122 are taken in consideration when the analysis was made.

Outcomes
It appeared that package level design quality indicators were not significantly affected by development method. [madeyski_06]

Possible motivational biases in estimation
No significant motivational biases were found in this article, only one small thing that could have been done is to explain why 16 teams decided to quit. Since these are not taken into the statistical evaluation, could it be some confidence issues in pair programming or something else perhaps? However, the author’s main research questions are answered and he investigated what he intended.

Pair-Programming Effect on Developers Productivity

Heiberg, Puus, Salumaa, Seeba
Springer-Verlag Berlin Heidelberg 2003

Article

Abstract
This paper gives an overview of a pair programming experiment designed to verify how pair programming affects programmer's technical productivity.

Experiment
The goal of the experiment is to find out how pair programming influences developers technical productivity. The hypothesis is as follows, "The programmers using pair programming technique have higher technical productivity than the programmers using traditional teamwork techniques." [heiberg_etal_03] In addition to the main hypothesis the authors also gathered information on satisfaction with the programming technique from the subjects.

The subjects were randomly divided into two groups, where in group "A" they used pair programming, and in group "B" they used traditional teamwork technique. The experiment was divided into two phases, the first two weeks subjects had two practice sessions, one session a week, and at the beginning of the second phase, each individual changed partner within their groups, again randomly. One more change was also made; group A switched programming technique with group B. Now group A was the one using the traditional approach, while group B used pair programming technique.
At the end of both phases students were asked to fill in a questionnaire that contained quantitative and qualitative questions about their attitude towards their working style, partner, result and experiment.

All the teams had to solve the same exercise – implementing a referee for a strategy game environment. In this environment two gaming programs (gamers) play a predefined strategy game with a rule checking program (referee) as a mediator. The students were responsible for creating the rule checking referees in phase 1, and the protocol implementations in the phase 2.

To be able to measure technical productivity a testing environment was developed by experiment team. Requirements were translated into automated test cases, which the authors used to evaluate quality of the solution. This test simulated two gamers, both with valid and invalid game situations according to the game protocol.

Results
The authors tried to answer two questions out of the data they collected,
1: How many solutions provided by pairs in given group satisfy any test cases at all?
2: What is the average percentage of satisfied test cases per pair on given group?

The authors conclude following:
- Phase 1, 1,7 times more pair programmers passed first test cases than non pair programmers
- Phase 1, the average number of passed test cases per pair was 1.9 times higher at pair programmers

- Phase 2, non pair programmers passed no test-cases

These results that represent the first week session shows that there were significant differences between pairs who used pair programming and the others, but after the first week the results were even. The authors conclude that this is probably due to the pair programming pairs used more time on design.

The questionnaire gave no significant differences between the test subjects, when trying to answer following questions:

1. What are the personalities of the programmers who perform well in pair programming setting?
2. What are the personalities of the programmers who perform well in pair programming, but badly in non pair programming setting?
3. What are the personalities of the programmers who perform well in non pair programming, but badly in pair programming setting?

Main research question

Q1	Q2
"Quality of the solution"	Passed test cases

The only quality measurement that is considered in this paper to conclude that one technique was "better" in quality is the number of passed test cases. Even though the authors fail to mention what kind of quality they are testing, passed test cases indicate the reliability and robustness of a program. The authors presented only a table in percentage, which answered only the two study questions they presented, instead also including a table with raw data, so one can conclude the results by themselves.

Population selection
The experiment was conducted in Estonia, at the University of Tartu, where the subjects were 110 students of the spring term 2002 Object-Oriented Programming course. Most of the students were first year students, with little experience with programming, but all had taken at least one semester with Java. Since the students are all relatively fresh to the programming world, they cannot be considered a proper population for the industry.

Outcomes
The outcomes were not of any significance; they could neither confirm nor reject anything out of the results. The experiment does not support the stated hypothesis, "the programmers using pair programming technique have higher technical productivity than the programmers using traditional teamwork techniques".

Possible motivational biases in estimation
The authors tipped a bit over to the pro pair programming technique, because of following reasons.
They never specified what traditional teamwork technique is, even though this could mean a number of things.

Even though the experiment does not support the stated hypothesis, the authors conclude that pair programming is not less productive, than non pair programming.

The first conclusions of the authors sound like this, “Let us assume that the experiment is internally valid” [heiberg_etal_03]. Why didn’t they bother to prove this or elaborate more on why it is or isn’t internally valid?

The authors also recommend pair programming as a developing technique because of the pair programming pair’s more test case passed in the training session and claim “If we could assume also some degree of external validity…” again, why assume if one can prove it?

Experimental Evaluation of Pair Programming

Nawrocki, Wojciechowski
12th European Software Control and Metrics Conference, UK, 2001

Article

Abstract
In this paper the authors compare two variants of programming, the PSP (Personal Software Process) [hump95] and pair programming (a part of eXtreme Programming). Four experiments are described that was performed at Poznan University of Technology.

Introduction and Experiment
The experiments conducted at Poznan were supposed to check the results obtained by Williams [williams_etal_00], and Nosek [nosek_98], and to increase the knowledge about pair programming.

The experiment was conducted among 4th year students, and the aim was to evaluate efficiency of pair programming, and compare pair programming with other practices of the XP with the PSP approach. The authors divided the subjects into three groups;

1. 6 programmers who used the PSP-approach suggested by Humphrey
2. 5 pairs who used XP approach (pair programming, experimentation and test-centered quality assurance, simple solution, risk minimization, keep moving)
3. 5 individuals using XP approach but without pair programming (in other words all the above except working in pairs)

To compare the results easier the author took 4 assignments presented by Humphrey and introduced them to the subjects. After every delivery the authors modified the programs that were turned in by the subjects, and introduced 3 errors, which the students had to solve again. This was done to insure that both students who were in pairs understood the program, and not was passive.

Results
First the authors compared the total development time average for each group, and here it shown that both groups using XP used less time (except for program number 4, where one student misunderstood the assignment and used more time). Also the authors presented standard deviation for the total time used to see whether some of the techniques showed consistency in development time. Only the pair programmers resulted in having least variation development time.

Another interesting attribute of a software process is programming efficiency measured in line of code (LOC) per hour per person. Here the results showed that XP practice without pair programming is the one most efficient in the matter of LOC per hour. The same group also wrote most LOC in average. In the LOC standard deviation graph, XP with pair programming was again the least variable group. Their programs showed the least amount of variation in lines of code.

At the end the last measurement was numbers of error uncovered during acceptance testing. The number should be equal to zero, but it wasn't. The graph shows a slight higher number for the individual group using PSP in the last program, otherwise it was fairly even.

Main research question

Q1	Q2
Total development time	Time used for each program
Programming efficiency	LOC pr. Hour
Software process efficiency indicator	Number of retransmissions after acceptance test

In the first efficiency measurement the individuals using PSP were told to write their first program on paper before coding it in a text editor, this can be the reason why they used more time on the first program than the other 2 groups. Also the last program was misunderstood by an individual using PSP, which also could alter the results. The programs were relatively small to perhaps test the real effect of XP, but it was done to compare the results relatively easily to the results that Nosek got.

Population selection
The population consisted of 21 4th year master students, all from Poznan University. All decided to program in C or C++, even though they could choose a variety of programming languages.

Outcomes
The results from the experiment shows following according to the authors:

- XP pair programming is less efficient then suggested by [williams_etal_00], and [nosek_98]. Only in one case (program 4) reduction of average development time was similar to what was reported by [nosek_98], but it is far away from what [Williams_etal_00] mentioned. Also one of the students misunderstood the assignment, which distorted the results a bit.
- Individual PSP seems less efficient than individuals using XP (without pair programming)
- Pair are more predictable regarding development time and average code size
- Experimentation and test-oriented thinking reduces development time (know as Test Driven Development)
- The number of re-submissions indicates that amount of rework for pairs is slightly smaller than for the individuals

Possible motivational biases in estimation
No significant biases were found, that were not mentioned in the text by the authors themselves. They tested both XP with and without pair programming to exclude the effect the other techniques (except pair programming) that XP presents, have on the results.

The case for Collaborative Programming

John T. Nosek
Communications of the ACM, 1998

Article

Introduction and experiment

The author starts the article by describing that previous research indicates that students programming in pairs outperform programming individuals.

The experiment is designed to answer four predictions, made based on previous results.
[nosek_98]

1. Programmers working in pairs will produce more readable and functional solutions to a programming problem than will programmers working alone.

2. Groups will take less time on average to solve the problem than individuals working alone.

3. Programmers working in pairs will express higher levels of confidence about their work and enjoyment of the process immediately following the problem solving session than will programmers working alone. (Positive feelings about the problem-solving process can affect performance. These feelings were assessed immediately following the problem-solving session by a two-item questionnaire.)

4. Programmers with more years of experience will perform better than programmers with fewer years of experience.

The subjects were 15 full time system programmers from a program trading firm (not mentioned which). The subjects were randomly assigned to groups and all subjects were given 45 minutes to solve the problem.

There are 5 measurements used in this experiment. They are readability, functionality, time used, confidence among programmers, and enjoyment. To evaluate readability subjects were asked to comment properly for each process in the program. Functionality was measured by achievements of the goal, confidence and enjoyment were rated by subjects answering a questionnaire right after the problem solving phase.

Results

The author used a t-test which is designed to compare two small samples. The pairs scored more on the readability, as well as functionality variables. Also the pairs used significantly less time to finish the task, with some variations in the standard deviation, but not very significant. Individuals used almost the maximum allowed time in mean. Pairs outperformed both confidence and enjoyment as well.

Main research question

Q1	Q2
Readability	Subjective measurement
Functionality	Subjective measurement
Time used	Measured in time, minutes used
Confidence	Subjective measurement
Enjoyment	Subjective measurement

All of these measurements except for one are subjective measurements, which mean that the author could have biased the results. Two of these measurements were subjectively rated by the programmers, confidence and enjoyment, and two of them were rated by the author, readability and functionality. Confidence is probably not just positive, as this is presented in the article; too much confidence can lead to overconfidence, which can result in the programmer taking on too much responsibility, or under estimates his developments skills. This is also just an assumption. Another question that can arise is why only 45 minutes were given? Since this can lead to unwanted results in behavior with the programmers.
Since the time boundary can lead to pressure, some of them might work better under pressure, and some might give in, and product of this can be a threat to internal validity. Not to mention that a program that is written in 45 minutes is relatively small to be used to compare and conclude two programming techniques. Most of the companies spend much more time to finish off a program.

Population selection
The population consisted of 15 professionals, who were divided into 2 groups, 5 pairs and 5 individuals.

Outcomes
Pair programmers scored better on every variable that was measured. They produced more readable code, offered better functionality, used less time to come up with the solution, and had more confidence in their work, as well as enjoyed working more.

Possible motivational biases in estimation
The author claims that pairs wrote better algorithms and code, but failed to explain how and why he claims this. Also he mentions that several studies previously conducted show that pair programming is outperforming the individual programmer, but he fails to mention which studies are these, how many are they, what was the criteria in these studies that indicate his claims? None of these claims are backed up with evidence. Since all variables used to measure the difference, only were positive for those who used pair programming, this can mean that the author used these variables on purpose, to present pair programming technique in a better light. Since the subject group was relatively small it can be a result of just coincidence.

An Empirical Comparison Between Pair Development and Software Inspection in Thailand

Phongpaibul, Boehm
ISESE, 2006

Article

Abstract

The objective of this study is to compare the commonalities and differences between pair development and software inspection as verification techniques in Thailand. One classroom experiment and one industry experiment were conducted. The development effort and effect of quality were investigated.

Introduction and experiment

This experiment was triggered by a request made on e-workshop where many researchers were very interested in investigating whether pair programming succeeds in its goals of providing same or improved benefits as inspections, with what cost, and in general whether the two practices were complementary and under what circumstances they each made the most sense [phongl_boehm_06].

The subjects in the undergraduate experiment were 95 Thai undergraduate students. Experiment was a part of the software engineering course the students were following. Student subjects were trained both in pair programming and inspection techniques. To avoid bias the authors clarified to the students that the objective of the experiment was not to explore which technique was better, but to understand the differences between both techniques. The authors also provided a design and code checklist to assist the students in detecting the defects. The second experiment consisted of 9 subjects, all professional developers from a multi-international company located in Thailand. The experiment was part of promoting the quality assurance program in the department.

Students were divided into 5-person teams based on their grades. All the teams used RUP (Rational Unified Process) as a developer process, and either pair programming or Fagan's inspection technique. The experiment lasted for 12 weeks, where the developments cycle composed of 3 phases; 4-week inception, 4-week elaboration and 4-week construction. There were total of 7 teams of pair programmers, and 7 teams using inspection. At the start there were 10 teams using pair programming and 9 teams using inspection, but due to different reasons they are not taken into account in this experiment. The industrial subjects were divided into 2 teams, with one project manager managing both teams.

The data was collected via data sheets, which looked like this;

Inspection data sheet;
Planning record, individual log, defect list, defect detail report, and inspection summary report

Pair programming data sheet;
Planning record, time sheet, individual defect list, and pair development summary report.

Hypothesis

This paper is focused on the difference between pair development and inspection in areas of the cost of process and the effects of quality. Total Development Cost (TDC) and component of Cost of Software Quality (CoSQ) are used for comparing the cost of process. The effects of quality are determined by number of problems found by TA, number of un-passed test cases, number of incomplete requirements, and total project score.

Main research question

Q1	Q2
Total Development Cost	Time in man hours
Distribution of Cost	Percentage of total cost
Defects found	Defects found by the TA (teacher assistant) after each phase
Un-passed test cases	Numbers of test cases the group didn't pass
Project score	Overall project score measured from all previous measurements combined

Total development time is measured in man-hours, to find out which method is more effective, according to calendar time. The second measurement Distribution of Cost, shows us how the two groups distributed their development, how much time was used in each phase of the project. For example pairs used more time in production, but almost no effort on rework and review (since this was done by the co-driver). After each phase in the project the students had to submit a package that contained an inception package, which contained vision document, a system requirement specification and a quality report. Out of these reports detects were counted by the teacher assistants. Un-passed test cases are measured by how many test cases each team did not pass, and the project score is an overall score calculated from the results of other metrics.

Population selection
The subjects in the undergraduate experiment were 95 Thai undergraduate students, and in the industrial experiment there were 9 professional programmers.

Outcomes
The result showed that pairs had 24 % less effort than the inspection group. There is a significant difference in the team's distribution of cost. The pairs used 50-60 % in production phase of the project, and almost no time in the rework phase, while the inspection group used less time in production but more in the rework phase. The TDC (total distribution cost) for the pair development teams is much lower than the TDC for inspection team, because the pairs discovered the defects in an early stage of the development, thus the less cost. Even though the inspection group had less un-passed cases, there was no significant difference in this metric, to conclude that one technique is better than the other. The same goes for the overall project score, which also showed no significant difference. In the industrial experiment the same metrics were used, but some of the outcomes were not consistent with the experiment done among students. TDC in the industrial setting were higher for pairs than for the inspection group, which is the opposite from what the students got.

Possible motivational biases in estimation
No significant biases in estimation were found. One thing that can be questioned, there is no explanation why the pairs were divided, not in pairs but in a group of 5 students, and how does this work with pair programming? According to the pair programming definition there should be a driver, and a co-driver which inspects the code etc. How did the rest 3 students fit in? Even if there were 2 pairs, what did the last student do? This could have produced faulty results.

Tracking Test First Pair Programming - An Experiment

Rostaher, Hericko
Springer-Verlag Berlin Heidelberg 2002

Article

Introduction and experiment
The experiments were conducted at FJA OdaTeam, which is a software development organization. The authors wanted to learn more about the practice of test-first programming, and create an exercise which would help additionally improve test-first habits. The programmers were asked to implement a simple system which was specified in advance and at the same time explicitly think of the three programming activities: testing, adding functionality and refactoring. The authors gathered quantitative information on how much time the programmers spend on each of the three activities.

The exact information they desired to get was the percentage of time spent for each activity, based on the experience level of the developer. How often pairs switch from the driver/navigator role, depending on the experience level, and how well the team performed pair programming versus solo programming. Here they used time spent to finish the program to measure the efficiency of pairs vs. solo. Hypothesis is that the pairs are not less productive than individuals doing the same tasks.

The groups were constructed by trying to get as many combinations of experience as possible. The experience level was divided into 4 levels, where level 1 was no experience at all, and level 4 was over 5 years of experience. All of the subjects used a program that helped the authors keep a track how much time is spent on each of the activity. The pairs also had to specify which person is doing what, e.g., person 1 is the driver now and we are testing. Person 2 is the driver now, we are adding new functionality. This was essential to measure to answer one of the research questions.

Main research question

Q1	Q2
Relative time spent (overall)	Time spent (hours, minutes)
Relative time spent for specific programming activity	Percentage of the total time
Switching between driver/navigator role	Count of how many switches were made

The main goal of this study was to learn more about the test-first practice, and see whether it had any connection with developer’s experience. The authors measured also the efficiency of pair programmers versus solo programmers, to see if there were any significant differences there. This was measured in time spent by the developers, to finish 3 user stories. Relative time spent for specific programming activity is measured by the authors to find out how much time a pair or individual uses on testing, adding new functionality and refactoring, with developer experience as a factor. This is presented as a percentage of the total time used. How often a pair switches roles is also measured to learn more about the development-“rhythm” of the pairs.

Population selection

The population consisted of 16 professionals, who all worked for OdaTeam. Some of these professionals had just finished school and started as a developer, while others had over 5 years of experience in the field.

Outcomes
The study revealed that 50-60 % of programming is spent in testing, as expected by the authors. Also it reveals that there is a big difference of testing time for inexperienced programmers, and the rest of the teams and individuals. They spent more time refactoring than others. The average time spent by both individuals and pairs is around 350 minutes, so it shows that for this task, pairs cost double as much. The results also showed us what a high frequency of switching pairs (more than 20 per day) and shot phases of uninterrupted activities means (5 min average) in numbers. [rostaher_her_02]

Possible motivational biases in estimation
There is some evidence that suggest a possible motivational bias, but even though there is evidence, the authors were unbiased towards the individual programmers. First of all, one of the authors is a co-founder of OdaTeam, which could affect the results. Second, the developers at OdaTeam uses XP practice daily, and were probably more confident to work in pairs than as individuals and in the description of the experiment the author write *"Defining appropriate stories was essential for the experiment to work. Good communication between business and technical members of the team is one of the strengths of XP. "*[rostaher_her_02] Stories are used as a XP practice and by this statement the authors show that some parts of the experiment are tailored to suit the XP practice, more than other practices. Since the main goal of the experiment was not to compare solo versus pair programming, none of these biases matter directly to the authors results.

Effects of Pair Programming at the Development Team Level An Experiment

Vanhanen, Lassenius
IEEE 2005

Article

Introduction

Different studies have reported that pairs produce better design with fewer defects in the code, in shorter elapsed time and more enjoyably without using more effort than solo programmers. Other kinds of benefits have also been reported. These results are contradicted in other studies and the reason for it is that mostly of those studies the subjects are instructed to solve a small task in isolation from other developers, e.g., non team context. This articles objective is to execute a well-planned experiment where co-located teams develop a larger piece of software using either pair programming or solo programming, and as a result get more data on the effect of pair programming [vanhanen_lass_05]. The authors also mention other studies done on the effect of pair programming, but discard these as well due to inconclusiveness of the results. They further state that LOC as a design quality metric or evaluating software quality based on a short demo are not reliable metrics.

Experiment

The subjects, master students on the 4th year, were divided into five teams of four developers. Three teams used pair programming, and two solo programming. The experiment was conducted as a non-compulsory course in the spring of 2004. The subjects were randomly selected into groups, on the basis of their programming skills, which were tested in an assignment before the experiment started. The project effort was fixed to 400 hours, i.e., 100th per person. Everyone had to spend at least 75% of the effort in co-located team session lasting 4-8 hours. The teams had to follow work practices such as iteration planning, collective ownership, version control, coding standard, continuous refactoring, unit testing, system testing, time reporting, defect reporting and documenting. [vanhanen_lass_05]

The authors derived a set of hypothesis to be studied in the context of comparing pair programming with solo programming. Most of the hypotheses are derived based on previous literature but also on personal experience. The hypotheses are listed under each quality measurement:

Productivity:
- H 1.1 the pair programming teams have lower uses case productivity than solo teams.
- H 1.2 Higher use-case complexity favors pairs as measured on use case productivity
- H 1.3 The pairs have higher project productivity than solo teams

Defects:
- H 2.1 After coding and unit testing the pair programming teams have fewer defects than the solo programming teams
- H 2.2 After system testing and bug fixing the pairs have fewer defects than solo teams

Design quality:
- H 3.1 The pairs create better software design than solo teams

Knowledge transfer:

- H 4.1 In the pair programming teams each developer understands more modules well than in the solo programming teams
- H 4.2 In the pair programming teams more developers understand each module well than in the solo teams

Enjoyment of work

- H 5.1 Pairs enjoy their work more than the solo programmers

Main research question

Q1	Q2
Productivity	Amount of work / effort spent
Defect rate	Defects counted pre-delivery, and post-delivery
Design quality	NCLOC per method to characterize the method size
Knowledge transfer	Questionnaire after delivered system (subjective measurement)
Enjoyment of work	Questionnaire after delivered system (subjective measurement)

Productivity is defined as the amount of work results divided by the effort spent. Project productivity is the sum of the implemented use cases divided by all effort spent for the project. Use case productivity is the inverse of the development effort (designing, coding, unit testing, bug fixing and documenting the code) of a use case [vanhanen_lass_05].
Defects are reported in after unit testing and corresponding fixing by the teams. A researcher counted the defects that were delivered with the system, based on a standardized system testing.
To measure design quality the authors used NCLOC (Non-Comment Lines of Code) per method to characterize the method size, i.e., the number of flows through a method to describe the method complexity, and the number of parameters to tell how much information is passed to the method. Reasonably small values for these metrics typically indicate good design [vanhanen_lass_05].
Knowledge of transfer is a subjective metric, and the authors used a questionnaire to determine the level of knowledge transfer within the team. This was also the case for determining the enjoyment of work metric.

Population selection
20 students on the 4th year of their studies participated in the studies. All of them rated their programming skills as average or above average comparing their skills with other students.

Outcomes

The pairs had 29 % lower project level productivity than the solo teams. The reason for lower productivity is that the pair programmers performed very poorly when implementing the first use cases. Thereafter the differences between solo and pairs were minimal. For use cases 1-10 pairs spent on average 44 % more effort than the solo teams.

Solo programmers discovered more pre-delivery defects than the pairs did. The pairs made 8% less defects to the code during development stage, but after delivery their system contained considerably more defects than the system delivered by the solo programming groups. The reason for this is that solo programmers discovered more defects pre-delivery and fixed them. The pairs had less careful attitude towards system testing due to relying too much on the peer review during development.
The method level metrics showed slightly better average values for pairs, but the overall results were inconclusive. One pair showed smallest proportion of complex methods, but the other team showed no differences from the solo teams.

Knowledge transfer questionnaire showed that pairs understood more of the overall system, and also understood more methods in depth than solo programmers. Even though the hypothesis is supported by the results, the results are statistical insignificant due to small sample size.

The last metric to be measured was enjoyment of the developers along the development process. On average the pairs enjoyed the work more, but the answers varied a lot. At the end more programmers preferred to work alone than in pairs (3 for, 4 against) and also 5 considered solo programming to be better for the overall success of this kind of project (2 answered pair programming).

H 1.1 – Refuted
H 1.2 – Refuted
H 1.3 – Refuted
H 2.1 – Supported
H 2.2 - Refuted
H 3.1 - Inconclusive
H 4.1 - Supported
H 4.2 - Supported
H 5.1 – Slightly supported

Possible motivational biases in estimation
There are not found any significant biases in the estimation.

Strengthening the Case for Pair Programming

Williams, Kessler, Cunningham, Jeffries
IEEE 2000

Article

Introduction
Prior to this article the authors claim that industry had working collaboratively and produced higher-quality software products in shorter amounts of time. But their evidence was anecdotal and subjective "It works" or "It feels right". To validate these claims, the authors have gathered quantitative evidence showing that pair programming improve software quality.

According to earlier studies found on the effect of pair programming, done among professionals [nosek_98], there are claims that pairs produced better algorithms and code, as well as use 60 % more minutes in time (effort), at return completing tasks 40 % faster. Other studies are also mentioned such as Constantine's study which showed that pairs coded faster and were freer of bugs than ever before ([williams_etal_00] referring to [constantine_95]).

Experiment
In 1999 41 senior software engineering students participated in a structured experiment. The purpose of the study was to validate quantitatively the anecdotal and qualitative pair programming results observed in industry. The students were divided into two groups, 13 students formed a solo programming group, while 28 students paired up, forming a group that will use pair programming. The collaborative pairs also had additional assignment to keep the workload equal between the two groups. The experiment compared cycle time, productivity, and quality between the two groups. Students recorded the time they spent on the project in a web-based tool. An impartial teaching assistant executed automated testing to analyze program quality.

In this study, mostly results acquired from previous studies are mentioned, even though the goal was to produce proof which back up these claims.

Main research question

Q1	**Q2**
Productivity	Time used to complete assignments
Development/Software quality"	Test cases passed
Effort	Time used multiplied by number of developers
Satisfaction	Subjective (Survey the 41 students)

The productivity was measured by how much time a group or individual used to complete a task. This was recorded by the subjects, and documented. The authors call it development quality but what they mean is probably a defect rate or robustness. How many test cases are passed after the submission of the program, if a defect is found the program will not pass the test case? Effort is measured by how much time is used multiplied by number of developers, which indicate effort used by the pairs are twice the time they used to complete the assignment. The subjects were surveyed about how much they enjoy collaborative programming six times. The authors presented other metrics which indicate pair's superiority towards solo programming, but there were no references to where this information came from, so it is not included as a quality attribute.

Population selection
The subjects were 41 senior students attending same class in '99. Senior student means 3rd year students, or advanced undergraduates.
Outcomes
For the first assignment, the pairs finished in a shorter elapsed time and produced better product than the individuals (what does better product mean, never elaborates this). On average pairs used 60 % more programmer hours to complete the assignment. After an adjustment period this 60 % dropped to 15 %. The students completed 4 assignments over 6 weeks, and the pairs passed more test cases than the individuals. Their results were also more consistent, while the individuals varied more (the authors should have measured standard deviation to further proof their claim).
The subjects were surveyed after the completion of the assignments whether they enjoyed collaborative programming, and 90 % stated that they did. How can a student which did not use collaborative programming in the experiment state that he/she enjoys collaborative programming?

Possible motivational biases in estimation
The authors fail to mention that [nosek_98] almost only used subjective measurements to justify his results. The one objective metric was time used to complete the assignment, which also was used to calculate effort. The other metrics such as readability, functionality, enjoyment and confidence were all subjective measurements.

Many claims are mentioned in this article without evidence supporting them, i.e., *"pair programmers we have observed agree that pair analysis and pair design are critical to success…Pairs consider many more possible solutions to a problem and converge more quickly on which solution to implement"[Williams_etal_00].* These are some of the claims, which have no form of evidence other than "someone said this".

Co-authors on this paper are Cunningham and Jeffries, which developed the concept XP, extreme programming which pair programming is a part of the XP practice. This alone is a very possible motivational bias to disturb the findings, not to mention the title of the article which has nothing objective about it. As the author of this article quote Gerald Weinberg's observations "The human eye has an almost infinite capacity for not seeing what it does not want to see…" it certainly is true about not having an objective view on pair programming, and experiments conducted about its effect, as far as the authors are concerned.

The authors fail to mention why the pair programming group was given extra assignments. They state it was to keep the workload equal between the two groups, but if one is studying the effect of a programming technique it does not matter if a group consists of 10 or 100 people, the workload should been the same. Otherwise the results can be distorted.

Exploring the Efficacy of Distributed Pair Programming

Baheti, Gehringer, Stotts
Springer-Verlag Berlin Heidelberg 2002

Article

Introduction
How effective is pair programming if the pairs are not physically next to each other? What if the programmers are geographically distributed? The authors compared software developed by virtual teams using distributed pair programming against collocated teams using pair programming and against virtual teams that did not employ distributed pair programming. By distributed teams, we mean that two members of the team (which may only consist of two people) synchronously collaborate on the same design or code, but from different locations [baheti_etal_02].

Experiment
The goal of the experiment was to investigate whether distributed pair programming is as efficient as collocated pair programming. The hypotheses are that the distributed teams who work synchronously will produce higher quality code (by quality the authors mean the grades obtained within the course at the end of the semester) than distributed teams that do not pair synchronously, as well be more productive (in terms of LOC / hr). Distributed teams will have comparable productivity and quality when compared to collocated teams.
Before the main experiment the authors did an initial experiment to see if distributed pair programming could done with one PC (the pairs communicate via audio and video, share desktop view, where one has the control and other only read rights). The experiment showed to be a success and the authors started on the main experiment. The experiment was conducted in a graduate class as a 5 week long project. The class consisted of 132 students, 34 of whom were distance learning. The teams were composed of 2-4 students, where the students could pick their teammates as they liked. At the end there were;

- Collocated team without pairs (9 groups)
- Collocated team with pairs (16 groups)
- Distributed team without pairs (8 groups)
- Distributed team with pairs (5 groups)

In order to record their progress, the students utilized an online tool called Bryce [bryce] to record their metrics. The metrics recorded were development time, lines of code and defects. Development time and defects were recorded for each phase. Since many of the students did not record the defects correctly the authors decided to not consider this metric in the analysis. There for the 2 metrics that were used were productivity in terms of LOC/hr, and quality in terms of the grades obtained by the students for the project.

Main research question

Q1	Q2
Productivity	Lines of code per hour
“Quality”	Average grade obtained by the group
Communication within the team	Subjective measurement

Cooperation within the team	Subjective measurement

The authors ended up measuring two metrics, one was productivity which they measured in terms of lines of code written per hour, and the last one is described as quality of the software, which they measured by calculating average grade obtained by the group. The group could get a maximum of 110 points, and the grades were decided according to some criteria which are not mentioned, so it is difficult to decide what kind of qualities of the software they meant. Also some of the grade was given after a half hour long demo to the teaching assistants, so it is reasonable to assume that oral skills played a part as well. At the end of the project, students answered a questionnaire on how the cooperation was within the team, and how the communication was within the team.

Population selection
The subjects were 132 students at a graduate class, where 34 of the students followed the class remotely (via internet etc.).
Outcomes
The distributed teams had slightly greater productivity compared to collocated teams, but due to high variance in the data for distributed groups it is not statistically significant. The grades obtained by the distributed teams were also slightly higher than for the collocated teams. As for the results from the questionnaire, the distributed pairs seemed to cooperate and communicate best.

Possible motivational biases in estimation
Some of the projects were sponsored by different sponsors (companies). Since it is not mentioned which groups did the sponsored projects, it can arise some motivational biases, if these students were, i.e., compensated to develop a software, or the results are distorted to present it in a better light to the sponsors.

Empirical Study on the Productivity of the Pair Programming

Canfora, Cimitile, Visaggio
Springer-Verlag Berin Heidelber 2005

Article

Introduction

Due to the lack of empirical studies in software engineering particularly pair programming, the authors realized an experiment to compare the productivity of a developer when performing pair programming and when working as a solo programmer. The cost of two developers working on the same computer is particularly in focus in this paper.

Experiment

The authors realized the experiment in accordance to the GQM paradigm, as follows: *Analyze* pair programming session *for the purpose of* evaluating *with respect to* its capability of reducing developing time of each single programmer *from the point of view of* the researchers *in the context of* students' groups with different degrees. The experiment is part of a larger research project.
Two hypotheses are given prior to the experiment, the H0 states "Pair programming does not affect the developing time spent by each programmer with respect to the solo programming" [canfora_etal_05], and the alternative hypothesis H1 states "Pair programming does affect the developing time spent by each programmer with respect to the solo programming" [canfora_etal_05].
The experiment was conducted on a class of 24 master students, who formed pairs themselves, and maintained within these pairs in both runs. The students recorded time used on an assignment, and wrote it down on a time sheet. The subjects got assignments that they did as pairs, and assignments that they did individually.

Main research question

Q1	**Q2**
Effort	Time used on a task
Predictability	Standard deviation derived from time used

There are two measurements that are used to determine the difference between a developer working solo, and in pairs. The first one is effort, which is easily calculated by measuring time used by the developers to complete a task. The second one is predictability which is derived from measuring time used, and calculating the standard deviation of the result. If the standard deviation is lower for one technique, it can indicate that one technique is more predictable than the other.

Population selection

The subjects were 24 master students who attended a high educational university course. Each subject performed both pair programming and solo programming alternatively.

Outcomes

The statistics show that developers spent 61% more time on the first run, working on the task individually then when working in pairs. The second run the individuals spent only 3 % more time. The authors try to explain this that the second task was more complicated and the pairs needed more time to establish strategies. Why this measurement is not mentioned as a result, i.e., pairs use more time on complex assignments? This can indicate evidence of motivational bias. Also pair programmers showed a lower standard deviation which can indicate that pair programming is more predictable.

Possible motivational biases in estimation
The authors mention a few previous works on the effect of pair programming, where they specifically mention only studies that have shown positive results towards pair programming, failing to mention studies that have shown negative results. To be specific the authors state *"In [baheti_etal_02] the authors investigate the performance of pair programming when the components of the pair are not co-located: they found that distributed pair programming seems to be comparable to collocated software development in terms of the productivity and quality. Productivity is measured as lines of code per hour."* [canfora_etal_05] The authors never mention what kind of quality aspect [baheti_etal_02] measured, because it is obviously not in best interest of pair programming's agenda. The quality [baheti_etal_02] measured was the grades obtained by the students at the end of the semester, after submission of the assignments and an oral presentation. This indicates a motivational bias towards the results, to shine pair programming in a better light. One of the authors acted as a mentor in the subject's class during the experiment.

Evaluating performances of pair designing in industry

Canfora, Cimitile, Garcia, Piattini, Visaggio
J. Syst. Software, Elsevier Inc. 2006

Article

Introduction
The authors claim at the start that there is initial empirical evidence that pair programming has positive effects on quality and overall delivery time, which is demonstrated in several controlled experiments. Because there is an asymmetry between design and coding, applying pair programming to the design phase might not produce the same benefits as those it produces in the development phase. In this paper authors reports a controlled experiment on pair programming, applied to the design phase and performed in a software company.

Experiment
The authors used a template based on the GQM (Goal Question Metric) to define the goal of the experiment.
Analyze the practice of **pair designing**, *for the purpose* of **evaluating**, *with respect to* **effort and quality**, *from the point of view* of **designers,** *in the context of* a group of professionals of software development **Information System** [canfora_etal_06].
The experiment was carried out in a software company called Soluziona. The company is in a high ranking position in the market of software professional services, and they have also reached level 3 maturity of CMMI (*Capability Maturity Model Integration). The CEO of the company wanted to verify if pair programming was as beneficial as claimed by many scientific reports, and whether it could be extended to the design phase. 18 volunteered professionals which belong to Soluziona were selected to participate in the experiment.*
The experiment was designed as follows, first a preparatory phase was run, where the subjects prepared themselves individually. The first run five pairs were formed, and 8 subjects worked individually. The second and last run the subjects switched roles, now there was 4 pairs, and 10 individuals. They were going to be measured in effort (time taken by the subjects to complete the assignments), and quality (each assignment rated from 0 to 1).
The hypotheses were;

- *H0a: there is no difference in the effort employed between pair and solo designing*
- *H1a: there is a difference in the effort employed between pair and solo designing*
- H0b: *there is no difference in the quality produced between pair and solo designing*
- H1b: *there is a difference in the quality produced between pair and solo designing*

Main research question

Q1	**Q2**
Effort	Time used on a task
Quality	Subjective score, ratio given for each task by the authors and two independent evaluators
Predictability	Analyses according to standard deviation numbers

Effort was measured as the time taken by the subjects (pairs or individuals) to complete the assignments. Subjects were asked to take note when they started the task and when they finished the task. Quality was never defined by the authors, but they measured it by introducing a subjective measurement. The scale composed of three values: 0 for incorrect, 0.5 for neither incorrect nor completely correct, and 1 for correct. This is the measurement that each assignment was measured to evaluate quality of the program. The last measurement arrived from the numbers that were presented by the authors, and is measured by calculating the standard deviation of time used. This indicates how predictable a programming technique are, thus smaller the standard deviation is thus more predictable is the technique.

Population selection
18 professionals who worked for the company (average 2 years), were selected to participate in this experiment. They were chosen to form a homogeneous sample in terms of skills and experience.

Outcomes
In the first run pairs used slightly less time than individuals, but in the second run the pairs used about 60 % more time. As for quality pairs scored slightly better score in the first run, but obtained a significantly higher score the second run. Also the pair programmer's standard deviation showed that pairs were more predictable than individuals.

- H0a hypothesis can only be rejected with regards to the data set corresponding the second run
- Pair designing and solo designing did not produce significant differences regarding the time spent on the single assignment
- The H0b hypothesis can only be rejected with regards to the data set corresponding the second run
- Pair designing produce significant higher quality than solo designing on each assignment

Possible motivational biases in estimation
No obvious motivational biases were found, but the authors seemed to be leaning more to the pair programming side rather than staying neutral. They conclude that pairs produce better quality in the assignments based on a simple measurement which they call quality. Since this company is CMMI certified at level 3, they should have intervened and perhaps tried to objectify the quality measurement more. The quality measurement is too simplified and subjective, there for can't be considered a reliable measurement.

Are reviews an Alternative to Pair Programming?

Müller
Kluwer Academic Publishers, 2004

Article

Introduction
Due to the ongoing discussions about whether pair programming really benefits as it is claimed, the author decided to investigate if a single developer can be assisted by an already known technique with which he produces the quality of pairs with only a fraction of the cost? Instead of estimating possible values for different parameters (increased speed, higher quality, etc) and thus delimiting the potential of pair programming over single developers, this study tries to seek an alternative technique with which a single developer produces code with almost the same quality but with lower cost [muller_04]. Inspections are a candidate but due to its high time consume and cost, the author decided to use reviews.

Experiment
This paper investigates with a controlled experiment if a single programmer equipped with an additional review phase can compete with pair programming in terms of code quality and development cost. The experiment was conducted as a part of a class at Karlsruhe University. The students had to be a part of the experiment to pass the course. They were given two tasks, one which involved calculation of the zero positions of a polynomial, and the other implement a shuffle puzzle. The study evaluated following null-hypotheses:

- *Reliability:* Pairs of developers do not produce more reliable code than single developers equipped with reviews.
- *Effort:* Pairs of developers are not more costly in person hours than single developers equipped with reviews.

The Review procedure (individual programmers)
Implementation was spilt into coding, reviewing and testing. During coding the individuals could only compile the code, but not execute it. After the coding was done the subject printed out their code, and handed it in for review to the subject’s pair programming partner. The review was anonymous, and the subjects did not review until both were finished. After the review was done the subjects got back the reviewed code, and tested it. Now they could execute the code, while testing it. When they felt the program was robust enough, they handed it in for an acceptance test, which consisted of 100 test cases. In order to complete the acceptance test successfully they had to pass 95 % of the tests, if they didn’t the code was returned, and the subjects had to fix the code before delivering it again. The acceptance test’s test cases derived from a large test which consisted of over 700,000 test cases. From these, the acceptance tests were selected randomly before the experiment started.
Each groups cost was also accumulated, and every student had to keep track of the time, and report it through Emacs’ add on.
Four hypotheses derived, and this study aims to verify them:

- H0 (reliability): Pair Programming <= Review
- H1 (reliability) Pair Programming > Review
- H0 (cost) Pair Programming > Review
- H1 (cost) Pair Programming > Review

The pair programming was straight forward. During implementation the pairs could compile and execute their program from very beginning.

Main research question

Q1	Q2
Reliability	Test cases passed
Cost	Pairs = 2 x (Read + Implementation + QA) Review = Read + Implementation + Review + QA

Reliability is determined by numbers of test cases passed. Since both teams used QA, and strict rules for passing through QA phase, both groups got very high scores. The cost is measured by the time used in each phase. Different formulas are applied depending which group is to be measured. Pairs = 2 x (Read + Implementation + QA), Review = Read + Implementation + Review + QA. Both formulas are also calculated without QA, to see calculate the cost of QA only. QA cycle-cost consist only of the time needed for the subjects to rework their deliveries.

Population selection
All 20 subjects were computer science undergraduate students who were on average in their fourth year of study. On average they had more than 6 years of programming experience, and some of them had even programmed in pairs before, and/or had experience with reviews. The subjects were split in 4 groups, all according to their skills. The aim was to obtain a general experience level within each group, so the lowest skilled had to pair up with the most skilled, second best with second lowest etc.

Outcomes
There is almost no difference between both groups concerning the reliability. Both groups have a mean about 99% on average. This is due to the QA phase; the solutions had to be 95 % correct to pass the acceptance test. If we remove QA's impact on the result, we get another set of results. Pair's average after the acceptance test is 57 %, where the reviewers got 43,8 %. On the large test the results were similar, 53 % pairs, and 46,6 % reviewers. But the authors claim that these numbers are not statistically significant due to a large variation in the data sample. The quality of the programs developed by single developers cannot be ascribed to the error finding capabilities of the reviews alone. However, the possible reason for the small difference in reliability could be a psychological effect because the developer's attitude to quality changes if he knows that somebody else is looking at his program.
The author measured cost for the whole task, for implementation, and for QA. For the whole task the differences are small, and vary a lot. There for are they not statistically significant. According to the whole task, single programmers are 5% cheaper. One statistical significant result was obtained during the implementation phase (excluding cost of QA). The review group used 357 minutes in average, where pairs used 407 minutes. Also pair's median was 409, and reviews median was 324. Even though this is a difference, these savings are lost during QA.
Answer to the two hypotheses,

- A pair of developers does not produce more reliable code than a single developer whose code was reviewed
- Although pairs of developers tend to cost more than a single developers equipped with reviews, the increased cost is too small to be seen in practice.

Possible motivational biases in estimation
No significant motivational biases are found.

Two controlled experiments concerning the comparison of pair programming to peer review

Muller
The Journals of Systems and Software, 2004

Article

Introduction
This paper reports on two controlled experiments comparing pair programming with single developers who are assisted by an additional anonymous peer code review phase. The experiments were conducted over two semesters, one in 2002 and the other in 2003 at the University of Karlsruhe. The subjects were 38 computer science students. Instead of comparing pair programming to solo programming this study aims at finding a technique by which a single developer produces similar program quality as programmer pairs do but with moderate cost.

Experiment
The experiment setup and settings are the same as described in "Are reviews an alternative to pair programming" [muller_04], which is analyzed prior to this article. This paper reports on the same experiment which is conducted one more time, only a year later. The reason for this is to compare the results obtained by the first experiment, and see whether a pattern occurs. In other words extend the results from the experiment done in 2002 by a thorough analysis of the combined data sets of both experiments. The experiment done in 2002 will be referred as Exp02, and the one done in 2003 will be referred as Exp03.

Main research question

Q1	**Q2**
Correctness (Reliability)	Test cases passed

Cost	Pairs = 2 x (Read + Implementation + QA)
	Review = Read + Implementation + Review + QA

Reliability is determined by numbers of test cases passed. Since both teams used QA, and strict rules for passing through QA phase, both groups got very high scores. The cost is measured by the time used in each phase. Different formulas are applied depending which group is to be measured. Pairs = 2 x (Read + Implementation + QA), Review = Read + Implementation + Review + QA. Both formulas are also calculated without QA, to see calculate the cost of QA only. QA cycle-cost consist only of the time needed for the subjects to rework their deliveries.

Population selection
All subjects were computer science undergraduate students who were on average in their fourth year of study. On average they had more than 6 years of programming experience, and some of them had even programmed in pairs before, and/or had experience with reviews. The subjects were split in 4 groups, all according to their skills. The aim was to obtain a general experience level within each

group, so the lowest skilled had to pair up with the most skilled, second best with second lowest etc. The same population was selected in Exp03.

Outcomes

Correctness

The mean values range between 93,3 % for pair programmers doing the first assignment and 98,3 % for Reviewers doing the second assignment. The reason that data points fall below the 95% exit criteria of the quality assurance phase is that the results of the large test and not the acceptance test are shown.

The correctness after the implementation varies between pairs and reviewers. The mean values for the groups vary between 36,1 for Reviewers and 59,6 for pairs. The standard variation also varies more than in the first experiment, between 29 and 35,6. The average level of correctness for pairs is 29 % higher than for reviewers (54,8 % versus 42,4%). Although there is a visible difference, the Mann-Whitney test that the author used did not show any statistical difference.

Summary of correctness is

1. The quality assurance phase served its purpose: programs of both groups have high and similar level of correctness.
2. The pairs developed programs with an observable high level of correctness.

Cost

Combining the two pair and the two review data sets, pairs (mean of 546,6 minutes) are on average 7% more expensive than reviews (mean of 511,8). However, the results are not statistical significant due to variance. In the cost for implementation, the pairs are on average 13 % more expensive (439,7 for pairs versus 389,2 for reviewers).

Cost for quality assurance

The result of Exp03 is surprising: the review group is cheaper than the pair programming group, although the single developers had to make up for a lower level of correctness after the implementation phase. The data of exp02 does not show this effect. In this case, the pairs are cheaper than single developers. However, both differences are insignificant. Further studies are sought to study whether pairs or single developers are more productive during quality assurance phase.

Possible motivational biases in estimation

No significant motivational biases are found.

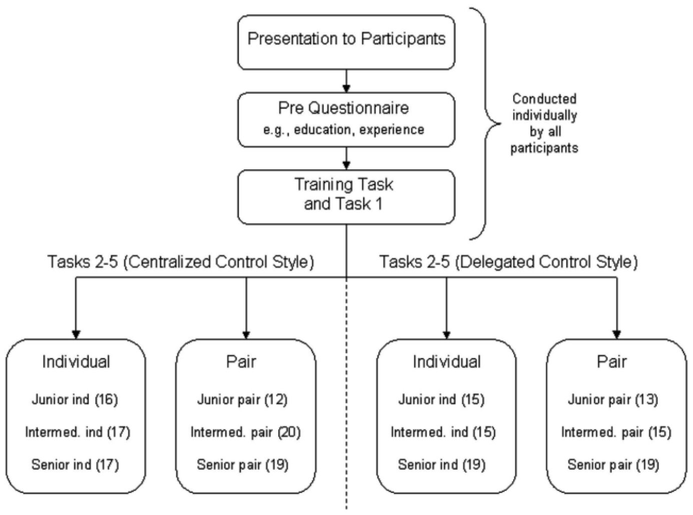
Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise

Arisholm, Gallis, Dybå, Sjøberg
IEEE 2007

Article

Introduction
A number of professionals with different programming experience were hired for one day, from different companies in Norway, Sweden and the UK to participate in a controlled experiment.
Previous literature suggests that pair programming significantly improves various measures of quality in the programs being developed. The authors of this paper wanted to find out *what is the effect regarding duration, effort and correctness of pair programming for various levels of system complexity and programmer expertise when performing change tasks?* [arisholm_etal_07]

Experiment
The dependent variables duration, effort and correctness represent one dimension of more general concepts such as time to market, costs and quality. The first part of the experiment was done individually. The 295 subjects were first introduced to the experiment, then they had to fill out a questionnaire regarding their education, experience etc, and at the end they had to solve a training task. A project manager from each company rated the subjects whether they were junior, intermediate or senior, the same way he would have rated them if they were about to be assigned to a "real" project. The subjects who had pair programming experience were assigned to the pair programming group (10 subjects had done pair programming earlier), and the rest got randomly assigned either as pairs or individuals. Further on, half of the pairs got assigned to change a centralized control system, and the rest worked on changes on a delegated control system. The same goes for individuals. Both groups were divided further to form junior, intermediate and senior both pairs and individuals. This can be seen on the figure below.



The main task (Coffee Machine) was based on two alternative Java systems that were designed and implemented with a centralized and delegated control design strategy. The task consisted of 4 incremental changes to the coffee machine (i.e., implement a coin return button; introduce a new drink choice etc).

Main research question

Q1	**Q2**
Duration	Elapsed time in minutes to complete change task
Effort	Total change effort in person minutes per subject
Correctness	A binary functional correctness score with value “1” if all change tasks were implemented correctly.

Duration is measured by each subject. Before starting on a task the subjects wrote down the current time, when they had completed the task they reported the total time in minutes. Non productive time between the tasks was not included. The total change effort in person minutes to complete the change task, the total effort for the pairs is multiplied by two. An automated test script extracted, compiled and tested the tasks. The score “correct” was given if no, or only cosmetic differences in the test case output and no additional serious logical errors were revealed by manual inspection of the source code.

Population selection
Total of 295 junior, intermediate and senior professionals Java consultants (99 individuals and 98 pairs) from 29 international consultancy companies in Norway, Sweden and the UK were hired for one day to participate in this controlled experiment on pair programming.

Outcomes
The effect of pair programming on duration
H01 (The duration to perform change tasks is equal for individuals and pairs): Insufficient support for the hypothesis.

H02 (The difference in the duration to perform change tasks for pairs versus individuals does not depend on system complexity): Rejected

H03 (The difference in the duration to perform change tasks for pairs versus individuals does not depend on the programmer expertise): Insufficient support for the hypothesis, not rejected.

The Effect of Pair Programming on Effort
H04 (The effort spent to perform change tasks is equal for individuals and pairs): Rejected, pairs used 84 % more effort than individuals.

H05 (The difference in the effort spent to perform change tasks for individuals and pairs does not depend on system complexity): Rejected, the pairs required 60% more effort for the easy task, and 112 % more effort for the more complex task, than the individuals.
H06 (The difference in the effort spent to perform change tasks for pairs versus individuals does not depend on programmer expertise): Inconclusive results, the hypothesis is not rejected.

The Effect of Pair Programming on Correctness

H07 (The correctness of the maintained programs is equal for individuals and pairs): Insufficient support for the hypothesis, not rejected.

H08 (The difference in the correctness of the maintained programs for pairs versus individuals does not depend on system complexity): Rejected, the effect of pair programming on correctness depends on system complexity.

H09 (The difference in the correctness of the maintained programs for pairs versus individuals does not depend on programmer expertise): Insufficient support for the hypothesis, not rejected.

Possible motivational biases in estimation
No significant motivational biases are found. Since the test lasted only one day, this could affect the term pair jelling, where pairs get to know more one another. Also the experiment was conducted in separate phases for the individuals and pairs (3 year between the experiments) which could to some point influence the results (new technologies, more training in specific programming techniques etc). This is all mentioned by the authors as well.

Appendix B - Disclaimer on collaborative work in master thesis

This is a disclaimer on partly collaborative work in master thesis together with Morten H. Bakken. 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

[al-kilidar_etal_05] Al-Kilidar, H., Parkin, P., Aurum, A., Jeffery, R.: Evaluation of effects of pair work on quality of designs. IEEE CS Press, 2005.

[arisholm_etal_07] Erik Arisholm, Hans Gallis, Tore Dybå, and Dag I.K. Sjøberg: "Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise", IEEE Computer Society, 2007

[aopmetric] Wroclaw University of Technology, e-Informatyka and Tigris developers: aopmetrics project http://aopmetrics.tigris.org/ , 2005

[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

[baheti_etal_02] Prashant Baheti, Edward Gehringer, and David Stotts: "Exploring the Efficacy of Distributed Pair Programming", Springer-Verlag Berlin, 2002

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

[beck_99] Beck, K. Extreme Programming Explained: Embrace Change, Addison Wesley, 1999.

[beck_03] Beck, K., Test-Driving development: by Example, Addison Wesley, 2003.

[bryce] A web-based software process analysis system - http://bryce.csc.ncsu.edu/tool/default.jsp

[canfora_etal_05] Gerardo Canfora, Aniello Cimitile, and Corrado Aaron Visaggio: "Empirical Study on the Productivity of the Pair Programming", Springer-Verlag Berlin, 2005

[canfora_etal_06] Gerardo Canfora, Aniello Cimitile, Felix Garcia, Mario Piattini, Corrado Aaron Visaggio: "Evaluating performances of pair designing in industry", Elsevier Inc. 2006

[constantine_95] L.L. Constantine, Constantine on Peopleware, Yourdon Press, 1995

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

[dict_prod_06] productivity. Dictionary.com. *WordNet® 3.0*. Princeton University. http://dictionary.reference.com/browse/productivity visited 7.1.2009

[dict_read_06] "readability," Dictionary.com. *WordNet® 3.0*. Princeton University. http://dictionary.reference.com/browse/readability. visited: 9.1. 2009.

[dict_enjoy_06] "enjoyment," Dictionary.com. *WordNet® 3.0*. Princeton University. http://dictionary.reference.com/browse/enjoyment . visited: 9.1. 2009.

[dybå_etal_07] Tore Dybå, Erik Arisholm, Dag I.K. Sjøberg, Jo E. Hannay, Forrester Shull: "Are Two Heads Better than One? On the Effectiveness of Pair-Programming", IEEE Software November/December 2007

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

[gloss_01] Webpage: http://www.esse.ou.edu/glossary_st.html visited 5.1.2009

[hannay_etal_09] Hannay, J.E., Dybå, T., Arisholma, E., Sjøberg D.I.K. The Effectiveness of Pair Programming: A Meta-Analysis. *Simula Research Laboratory, Department of Software Engineering*, 2009.

[heiberg_etal_03] Sven Heiberg, Uuno Puus, Priit Salumaa, and Asko Seeba: "Pair-Programming Effect on Developers Productivity", Springer-Verlag Berlin, 2003

[hulkko_abra_05]	Hulkko, H., Abrahamsson, P.: A multiple case study on the impact of pair programming on product quality. Software Engineering (ICSE'05), 2005.
[hum89]	W. Humphrey, "Managing the Software Process", Addison-Wesley 1989
[hump95]	W.S. Humphrey, A discipline for Software Engineering, Addison-Wesley, Reading, 1995
[inf5180]	Pfahl, Dietmar: Course inf5180 – "Software product and process improvement in system development" slide part 05, p. 47. Fall 2008 http://www.uio.no/studier/emner/matnat/ifi/INF5180/v08/undervisningsmateriale/Lecture%20Notes/INF5180_P05bw-dp.pdf
[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
[lui_chan_03]	Lui, K., Chan, K.: When does a pair outperform two individuals? In: Proceedings of XP 2003 LNCS. Springer-Verlag.
[madeyski_06]	Lech Madeyski: "The Impact of Pair Programming and Test-Driven Development on Package Dependencies in Object-Oriented Design — An Experiment" Springer-Verlag Berlin, 2006
[martin_94]	Martin, R.C. – OO Design Quality Metrics, an Analysis of Dependencies, 1994
[martin_04]	Martin, R.C. – Agile Software Development, Principles, Patterns and Practices. Prentice Hall, 2004
[mayo_08]	Webpage: www.mayomedicallaboratories.com/about/quality/framework/glossary.html visited 6.1.2009
[mcdowell_etal_02]	McDowell, C., Werner, L., Bullock, H., Fernald, J.: The effects of pair-programming on performance in an introductory programming course. ACM, Cincinnati, KY, USA. 2002
[muller_04]	Matthias M. Müller: "Are Reviews an Alternative to Pair Programming? " Kluwer Academic Publishers, 2004
[muller_05]	Matthias M. Müller:"Two controlled experiments concerning the comparison of pair programming to peer review", Elsevier Inc. 2005
[muller_06]	Muller, M.: A preliminary study on the impact of a pair design phase on pair programming and solo programming. Information and Software Technology 48, 2006.
[nawrocki_woj_01]	Jerzy Nawrocki, Adam Wojciechowski: "Experimental Evaluation of Pair Programming", KBN, 2001
[nett_0607]	http://www.nettavisen.no/bil/article1097164.ece
[nosek_98]	John T. Nosek: "The Case for Collaborative Programming", Communications of the ACM, 1998
[phongl_boehm_06]	Monvorath Phongpaibul, Barry Boehm: "An Empirical Comparison Between Pair Development and Software Inspection in Thailand", ISESE'06 ACM, 2006
[rel_wiki]	Wepage: http://en.wikipedia.org/wiki/Reliability visited 1.11.2008

[rostaher_her_02] Matevz Rostaher and Marjan Hericko: “Tracking Test First Pair Programming – An Experiment” Springer-Verlag Berlin, 2002

[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.

[seg_07] SEG and Department of Computer Science, University of Durham, “Guidelines for performing Systematic Literature Reviews in Software Engineering”, 9.July 2007

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

[succi_etal_02] Succi, G., Marchesi, M., Pedrycz, W., Williams, L.: Preliminary analysis of the effects of pair programming on job satisfaction. Fourth International Conference on eXtreme Programming and Agile Processes in Software engineering, 2002.

[vanhanen_lass_05] Jari Vanhanen and Casper Lassenius: “Effects of Pair Programming at the Development Team Level: An Experiment” IEEE, 2005

[williams_etal_00] Laurie Williams, Robert R. Kessler, Ward Cunningham, Ron Jeffries: “Strengthening the Case for Pair Programming”, IEEE Software, 2000

[williams_03] L. Williams and R. Kessler, Pair programming Illuminated, Addison-Wesley, 2003

[williams_kessler_00] Williams, L., Kessler, R.R.: The Effects of ‘‘Pair Pressure’’ and ‘‘Pair-Learning’’ on software engineering education. IEEE CS Press, 2000.

[wilson_etal_93] Wilson, J., Hoskin, N., Nosek, J.: The benefits of collaboration for student programmers. 1993, 24th SIGCSE Technical Symposium on Computer Science Education

[xu_rajlich_06] Shaochun Xu, Vaclav Rajlich: “Empirical Validation of Test-Driven Pair Programming in Game Development” IEEE, 2006