Universiteit Leiden

ICT in Business

Software Quality Metrics in Practice

Name: Antonis Passos
Student-no: s2085593

Date: 5/8/2019

1st supervisor: Dr. W. Heijstek

2nd supervisor: Dr. C. J. Stettina

Acknowledgements

It is meaningful to express my gratitude to the participants of the interviews/surveys for spending part of their valuable time in order to share their experiences with me. Through their valuable contribution I managed to form this research. I am grateful for your effort.

I would like to thank my first supervisor, Mr. Werner Heijstek for his valuable guidance, ideas and recommendations from the very first moment. I would also like to thank my second supervisor, Mr. Christoph Stettina for his feedback and support.

Executive Summary

Software quality is of significant importance while delivering software nowadays. Software products' complexity and size is growing, similar to customers' demand for the end product. Hence, software quality is of high necessity in order to deliver well maintainable products.

By utilizing software metrics, software industry has the opportunity to "measure" the development process, resulting in high quality software systems. Multiple metrics that capture different aspects of quality have been proposed while their relevance is increasing. However, due to multiple reasons development teams bypass software quality practices resulting in poor quality software products. This research provides a brief view of software quality, software quality metrics and tools that are used in order to predict and measure the quality factors of a software.

Method: Interview and survey questions were constructed in order to identify the challenges and the needs from the development teams. Secondary data were also analyzed in order to shape the conclusion.

Results: The awareness regarding software quality and the use of metrics is increasing. A lot of software teams are making use of tools and metrics in order to gain valuable insights regarding the quality of their products. Working with the Agile methodology makes the metrics integration easier due to the iterating processes that forces the teams for continuous improvements through tests and insights. Quality metrics such as lines of code, unit test coverage, cyclomatic complexity and code duplication are amongst the most popular in the organizations. Other than that, there is the use of multiple tools that provide insights regarding software quality. However, through our research we identified that companies do not provide the appropriate training to their development teams regarding the use of the metric tools. Moreover, we gathered some improvements that our resource people would like to see at the tools they use, such as having better integration with the development environment that they use.

Conclusion: Software definitions such as "software quality" should be operationalized in teams so that everyone has the same belief of such important terms. Different interpretations by people regarding these entities may lead to difficulties in understanding the significance of software quality metrics usage. Moreover, organizational strategy regarding software quality should be implemented and better communicated to the teams in order to be aware of the software quality principles.

Table of contents

List of Figures

List of Tables

1.Introduction

1.1 Background

Software development and engineering is a fast-paced process. The software industry is considered to play a significant role as far as the economic growth is concerned. However, software products have become more complex resulting in multiple challenges for software companies when it is time to deliver high-quality software and hence, they strive to achieve customer satisfaction (Elgebeely, 2013). The complexity of software results in a boost in time and effort needed to understand how these systems maintain their components and extend their functionality. Therefore, this evolution that software industry faces causes issues to both the development and the maintenance process (Ghanam et al., 2008).

Accordingly to the mentioned above importance of software demand, having high software quality is an essential characteristic for any organization. Software quality is focused on satisfying customers' needs regarding with the software product (Sfetsos et al., 2010) and as an attribute it needs to be built-in during the development process (Kasurinen et al., 2012). System and software quality has been a significant area of attention in the computer science field. Its importance is steadily increasing as software becomes a vital part in everyday operations.

However, software systems need to be maintainable. According to the Institute of Electrical and Electronics Engineers (IEEE,1993), software maintainability is defined as the ease in which a software system can be modified to correct faults, improve performance and adapt to a changed environment. Thus, maintainability is the main factor influencing the time that a software can adapt or face new circumstances and changes. Consequently, it is an important attribute of software quality.

Moreover, in order to meet their strict deadlines and produce more features multiple companies are putting the pressure on the development teams. This results in neglecting a large number of defects in the code, architectural mistakes and ignoring documentations (Elgebeely, 2013). Consequently, by considering coding and the whole development work to be an overriding priority, most development teams bypass software quality practices resulting in poor quality software products.

Bouwers et al. (2013) claim that developers require feedback to track the maintainability of the software systems during their development. This feedback can be derived through software quality metrics which provide measurements to ensure the quality of the processes of the software development life cycle. However, Czewronka et al. (2015) highlight in their research that by focusing on software quality feedback the software development process slows down.

Nonetheless, through the utilization of metrics, software industry has the opportunity to quantify the development and the maintenance of software resulting in an improved quality of software systems. Various empirical studies consider metric measurements as the primary indicator of prediction and software maintenance. Their relevance is steadily increasing, becoming important for software management while their values are useful in order to determine the complexity and maintainability of the code (Chawla et al. 2013). Furthermore, software quality metrics provide an effective way to manage the quality of the processes of the software development life cycle.

Bijlsma et al. (2011) argue that the benefits of software quality are realized on the long term when the system is in operation. Metric data are able to provide quick feedback and according to Briand et al. (1999) using early quality indicators based on objective empirical evidence is a realistic objective. Rawat et al. (2012) claim that software quality metrics improve software quality while its importance is expected to increase in the coming years. As a result, software quality metrics are a powerful tool that needs to be used with care.

1.2 Research objective

The scope of this research is to contribute to the field of software quality and the use of software quality metrics and provide an overview on how these can be integrated in the development process without being burdensome for the development teams. On a scientific and research level, there are no significant studies conducted on the relation of this aspect. Hence, this study will attempt to answer questions based on the issues mentioned above. To support this understanding, this research aims to conduct a literature review and a combination of survey/semi-structured interviews and secondary data derived from interviews with experts in the software development field. The overall research objective is to identify the challenges and needs from the development teams and of course to determine whether and how these can benefit from the use of software quality metrics.

1.3 Research relevance

This research has relevance to software quality and the use of software quality metrics during the development process. Moreover, there are no similar studies conducted covering the topics of software quality and metrics. Consequently, the scope of this study is to contribute to the use of quality metrics during the development process, highlight the challenges and provide ways on how these can be integrated in the development process without affecting negatively the productivity of the team.

1.4 Research questions

Considering the above, the formulation of the main research question is as follows:

- How can software quality metrics be integrated in the development process in an unobtrusive manner? (RQ)

Research sub questions:

- How do organizations measure their software quality? (RSQ1)
- What are found to be the most effective software quality metrics? (RSQ2)
- How does the development process impact the software quality metrics integration? (RSQ3)

1.5 Research scope

The present research thesis will focus on software quality metrics. Consequently, the purpose of this study is to identify the challenges in practice.

In our research we will try to identify, evaluate and interpret the literature relevant to the topic of "software quality metrics". As a result, a literature review will be conducted. Furthermore, both survey and semi-structured interviews with experts involved with software delivery will be scheduled in order to identify the challenges in practice. There will also be use of secondary data.

Through the literature review, we will analyze the existing evidence in order to identify the different software quality metrics used during the development processes. What is more, we will introduce the potential of software quality metrics and its relevance of delivering quality software. Last but not least, the surveys/interviews and the secondary data will help us identify the challenges and the needs from the experts' perspective and finally discuss and propose our findings in combination with the literature review. In conclusion, we will have the opportunity to verify the processes regarding software quality in the software community.

1.6 Thesis overview

In the following table (Table 1) a brief overview of this thesis is presented.

Chapter	Content
1	The first chapter will provide an introduction of this research, providing the reader with information regarding the research objective, research relevance the research questions and finally with the research scope
2	In the second chapter, the reader will be exposed to the literature review of the study in order to familiarize with the concepts relevant to the research.
3	Chapter three presents the research approach followed.
4	In the fourth chapter, we will present the results of our survey, the semi-structured interviews and the secondary data.
5	In chapter five, we will analyze and discuss our findings by answering the research questions. This will result from analyzing the findings both the literature and the empirical data gathered from the semi-structured interviews, surveys and the secondary data.
6	The last chapter will state the conclusions and recommendations on how software quality metrics can be integrated in the development process.

Table 1: Thesis Overview

2. Literature review and related work

The following chapter contains the theory in order to support this research. In order to provide answers to the study's research questions it requires further knowledge on the topics relevant to the subject. More specifically, software quality is investigated through the software engineering literature. Furthermore, there will be an introduction to software quality metrics as a means of measurement of software quality

2.1 Software development processes

According to Modal et al. (2012), software development life cycle (SDLC) is the process of developing and maintaining a software system which is compiled by models and frameworks in order to plan and maintain the whole development process. The SDLC contains different phases such as analysis, system design, programming and testing, installation and maintenance. There are different models of software development processes. Thus, each model handles the software development process in its own way. Purcell (2007) presented in his research the most well-known models among which were: Waterfall, Iterative, Spiral, V-model, Big Bang Model, Rapid Application Development Model, Prototyping, Agile and Extreme Programming.

No matter the chosen model, the SDLC defines a methodology in order to improve the software quality and the overall development process. Hence, in order to meet the fast-changing user requirements, development teams have to follow correctly the SDLC steps in order to create high-quality systems.

2.2 Software systems

After getting familiar with the SDLC, it is crucial to provide a common understanding of the 'software systems' meaning. According to McDermid (2013) a software system is an aggregation of programs, documentation and operations procedures.

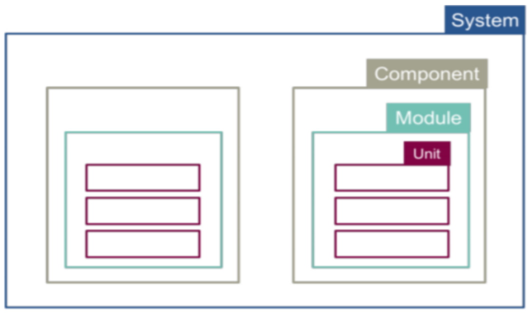


Figure 1: Software Systems' modules (Visser et al. (SIG), 2018)

Moreover, a software system consists of system components, modules and units. According to Visser et al. (2018) :

- Components: They are a subdivision of a system which contains the source code modules grouped together. The grouping is based in either functional or technical aspects.
- Modules: Modules correspond to different files. Depending on the programming language used these files correspond to classes.
- Units: Represent the smallest piece of code within a system.

2.3 Software maintainability

It is a fact that producing easy maintainable software, potentially saves large costs especially today that the control and the management of the changes in systems is one of the biggest challenges that needs to be faced (Penny, 2003). Hence, software maintainability is widely accepted as the ease with which a software system or component can be modified in order to prevent faults, improve performance and adapt to a continuously changing environment (IEEE, 1993)

Software maintainability is an important software quality attribute and this can be derived from the research of Boehm et al. (1976). In particular, measuring and monitoring software maintainability is a crucial aspect in software development. As a result, maintainability as the main factor influencing the time that a software can adapt or face new circumstances and changes is an important attribute of software quality. Software maintenance consumes 40%-80% of the total software costs rendering it as the most important phase of the SDLC (Glass, 2002). As a result, it is crucial that maintainability can be measured so that development teams can act upon.

Olatunji et al. (2010) classified software maintenance into four types:

- Corrective: fixings bugs/defects to a program.
- Adaptive: refers to modifications in order to adapt to changes.
- Perfective: enhancements in order to make the product better, faster, smaller and better documented.
- Preventive: the process in order to prevent malfunctions and improve maintainability.

Furthermore, maintainability as concept is difficult to quantify. Various models have been proposed the last four decades. One of the most recent model places maintainability as the top attribute influencing internal quality and it subdivides it into the following characteristics (Heitlager et al. 2007) :

Name	Description
Analyzability	How easy or difficult is it to diagnose the system for deficiencies or to identify the parts that need to be modified?
Changeability	How easy or difficult is it to make adaptations to the system?
Stability	How easy or difficult is it to keep the system in a consistent state during modification?

Testability	How easy or difficult is it to test the system after modification?
Maintainability conformance	How easy or difficult is it for the system to comply with standards or conventions regarding maintainability?

Table 2: Characteristics of Maintainability (Heitlager et al. 2007)

A software maintainability model like the one proposed by Heitlager et al. (2007 enables organizations to predict the maintainability of their systems and better manage their resources while adopting a defensive design (Oman et al. 1994). Consequently, this can help to reduce the maintenance effort and cost spent on a software project resulting in improved software quality -which will be analyzed in the next chapter-. Last but not least, a maintainable software with a high-quality code is more likely to have improved reliability, performance and security.

2.4 Software quality

Quality is always an issue while developing software. Software market is increasing resulting into customers who are expecting a higher quality of products. Thus, the growing customer needs in combination with the complexity and the size of the software products has put a constant pressure into delivering quality products within the time schedule and with less effort. The increase in expectations in the software market, leads companies to continuously invest a significant amount of money, time and effort in order to improve their software quality (Kitchenham 1996). Imreh & Raisinghani (2011) argue that quality and the emphasis on it are attributes that any organization should have in order to be successful. Therefore, in order to increase productivity and customer satisfaction, organizations need to define, measure, understand, analyze and control software quality during the software development.

According to the Institute of Electrical and Electronics Engineers (IEEE), quality is the degree to which a system meets the specified requirements and customers'/users' needs or expectations (IEEE, 1998). Quality in the software development process is focused on satisfying customers' needs regarding the software product (Sfetsos et al. 2010) and as an attribute it needs to be built-in during the development process (Kasurinen et al., 2012). Building high-quality systems result in business benefits such as higher customer satisfaction, improved delivery, predictability and better system performance (Leffingwell et al., 2016).

To conclude, one can say: Software quality means the ability of the end product to fulfil or exceed user's expectations. Franca and Soares (2015) highlight in their research the importance of software quality in the development process. Moreover, they make a distinction between the factors that affect quality. Last but not least, according to Beck (2000), external quality as an attribute is measured by the customers, while internal quality is measured by the developers.

2.5 Literature - Software Quality Models

Software quality is refined by several models and standards through a set of characteristics and sub-characteristics. The scope of these models is to provide a basis for understanding and specifying quality requirements and thus assess the quality of software (Miguel et al. 2014). The stakeholders of a product need to be satisfied and software quality models provide the understanding in order to set the quality goals of a software product. They consist of measurable characteristics and sub-characteristics in order to specify and measure the software product quality. During the next sub-chapters, we will refer to the most important software quality models.

2.5.1 ISO/IEC 9126 Quality Model

The International Organization for Standardizations and the International Electrotechnical Commission (ISO) with their model 9126 (ISO, 2001), evaluates software quality and defines six product quality characteristics which have to be met by software products in order to achieve high quality standards.

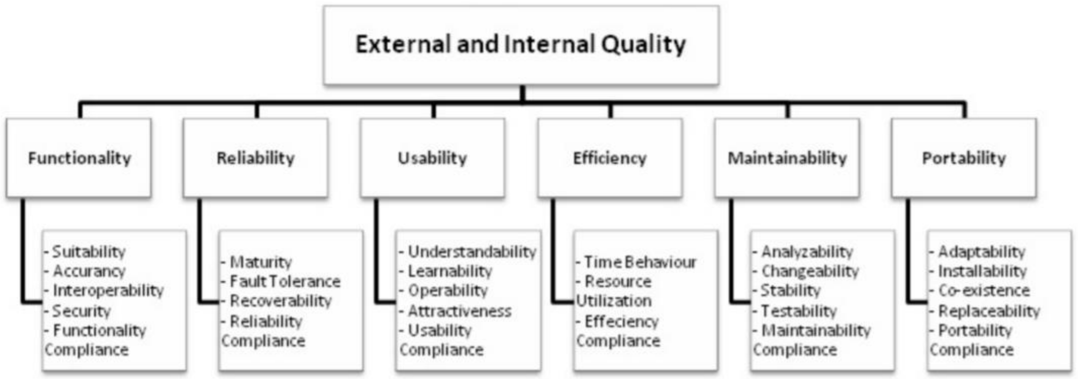


Figure 2: Notions of internal and external software product quality (ISO/IEC 9126-1)

Apart from maintainability which was analyzed in chapter 2.2, this model defines 5 more characteristics as shown in Figure 1. According to the model, maintainability is a characteristic related to the internal quality while system's functionality, reliability, usability, efficiency and portability are related with the external quality. In essence, this model distinguishes software product quality into three views

- Internal Quality: it is related with the functions of the system that can be measured without executing it (construction phase).
- External Quality: it is related with the functions of the system that can be monitored during its execution (testing phase).
- Quality in use: it is related with the functions of the system experienced during its operations and maintenance.

All of these three views are interrelated. As a result, the internal quality influences the external quality which lastly impacts the overall quality in use. The quality in use focuses on customers while both the internal and external quality emphasizes on developer's viewpoint.

2.5.2 SQuaRE ISO 25010 Quality Model

The Software Product Quality Requirements and Evaluation (SQuaRE- ISO 25010), introduced by ISO in 2011 as the next generation of software quality standard and as a replacement of the mentioned before ISO 9126. This product quality model adds two more quality characteristics at the ISO 9126, which are the security and the compatibility attributes. Hence, the ISO 25010 consists of eight characteristics and 31 sub-characteristics as shown in Figure 2 (Franca et al. 2015).

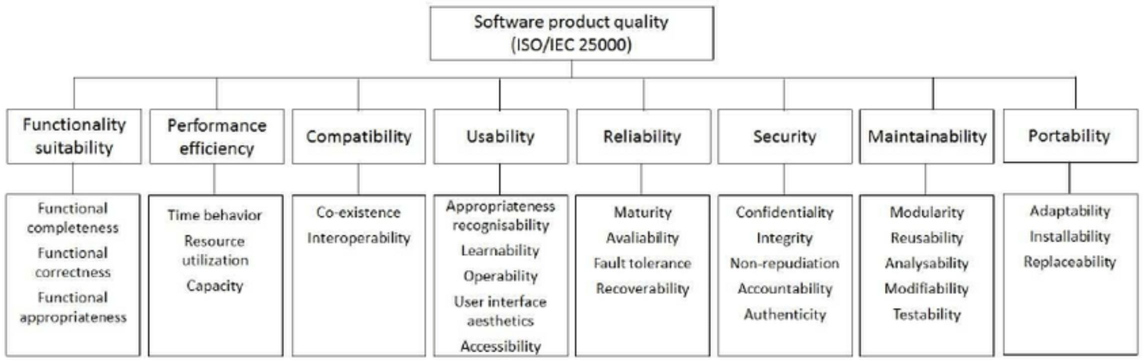


Figure 3: Notions of internal and external software product quality (ISO 25010)

The 2 extra characteristics and 31 sub-characteristics of this model render this model as a more complete and comprehensive version of the ISO 9126. Consequently, ISO 25010 is a great addition for the enterprise world and for software teams who want a framework in order to define the quality of their software. By breaking down the eight quality characteristics into sub-characteristics, developers can define software measurements that make sense for their projects.

2.5.3 SIG Quality Model

In order to operationalize the quality attributes derived from the ISO/IEC 9126 model the Software Improvement Group (SIG) developed a quality model for measuring the maintainability of the production code. This model maps a selection of code metrics regarding the maintainability attribute and further categorize it into sub-characteristics (Heitlager et al., 2007). It introduces another level below the framework by ISO/IEC 9126 that consists of system properties as shown in Figure 2.

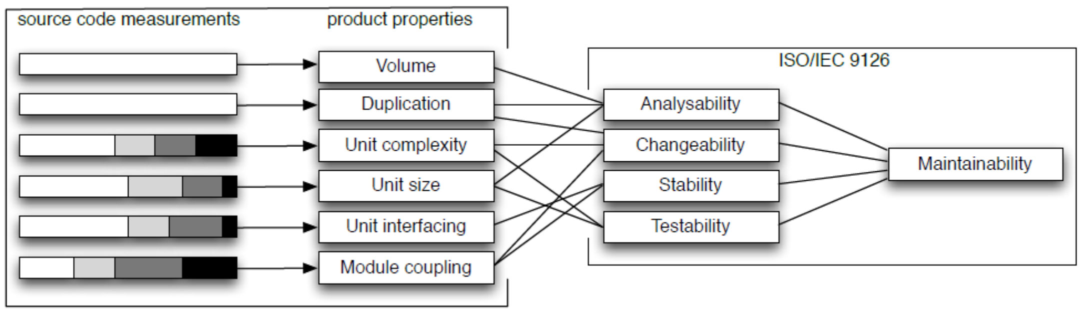


Figure 4: Quality Model by SIG (image from Luijten et. al 2010)

This model measures maintainability on the aspect of internal product quality mentioned in the previous chapter. As a result, it can be applied in software products in the construction phase. Therefore, the properties of a product can influence its maintainability and its characteristics which according to the model and to Visser (2015) these are:

Volume: Keeping the source code concise makes the product easily maintainable. A larger system requires more resources in order to maintain leading to lower analyzability. In order to measure volume, many different measurements have been proposed. The Line of Code (LOC) is the most well-known which counts the non-comment and non-blank number of lines of the source code of a system. According to Sato et al. (2007), classes with higher LOC are more error prone. However, except from LOC there are some supplementary estimates. Measuring functional size by counting the database tables, screens or input choice makes sense for some systems. Nevertheless, these measures do not have an impact on general volume but on functional size while they are not easy to calculate (Heitlager et al. 2007).

Duplication: Avoiding multiple occurrences of the same code makes the product easier to maintain. A system with excessive duplication is larger than it needs to. There are many techniques in order to measure the duplication issues, also called clone detection (Baker, 1995). The metrics used in this extend allow root cause analysis by tracking down the duplication issues

Unit size: Increasing unit size makes the software product harder to analyze and hence to maintain it. Lines of code per unit can be used to measure unit size in order to improve systems' maintainability.

Unit interfacing: Units with large interfaces are deemed to be harder to maintain. The size of these units can be quantified as the number of parameters, known as formal arguments. (Visser, 2015)

Unit complexity: The complexity of the source code has to do with its intricacy. High unit complexity results in a difficult to analyze and test software product. Cyclomatic complexity per unit and then summation of all unit complexities provides insights regarding the complexity of the entire system (Heitlager, 2007).

Module coupling: Strongly coupled with other modules code is harder to maintain. By using the module definition, we mean groups of multiple units such as classes.

2.6 Software measurement

After referring to the quality models it is remarkable to operationalize software measurement. In general, measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in order to describe them according to clearly defined rules (Finkelstein, 1984). Software measurement is the process of quantifying software attributes in order to better understand the effectiveness of the methods and the tools used and possibly customize them in order to achieve the project's goals. In software development process by collecting and measuring the necessary data, organizations are provided with valuable information that might optimize decision making on behalf of productivity and quality (Fenton, 1998).

In order to measure their software, organizations need to spend multiple resources, budget and technical expertise. However, software measurement is meaningful since it can predict software costs, software size and maintenance in the early stages of the development and thus it helps with managing and controlling the SDLC (Zuse, 1998). Development teams can use software measurements in order to derive information for fault tolerance, testability of requirements and the quality of the software product (Lundberg et al., 2005). Hence, by deriving information about the software quality through measurements from the software product, each organization is provided with the ability to negotiate with the customer regarding the software requirements (Fenton, 1998). Nevertheless, Hendriks et al. (2000) highlight in their research that evaluating quality in an unambiguous way is something difficult.

2.6.1 Software quality metrics

Software quality metrics are the source of information through which a developer can make decisions regarding the software developed. The organizations that develop software solutions need measurements in order to check whether their product meets the characteristics stated by the quality models mentioned in the previous chapters. According to ISO 1061 (1998) metrics used in the software development process use software data as an input while the output is a numerical value that is interrelated with software quality. Hence, through the use of metrics, a software developer is able to measure and predict the necessary software resources for a project.

Metrics provide development teams with a quantitative way to get insights for the quality of the internal attributes of a product (Punia et al. 2016). Furthermore, they enrich development teams with visibility and insights about what they do and how well they do it (Eeger, 2012). According to Phalke et al. (2014), metrics are the numerical value of a software process through which a development team can predict faults. As a result, people involved with software delivery, can find existing defects while they can prevent their team from facing defects in the future.

2.6.2 Software quality metrics usage benefits

The selection of metrics should assist development teams for better results. Consequently, software metrics should be simple and give experts the ability to define and attain their objectives through measurements. In that way, process production planning is supported, monitoring is optimized and the system's maintenance is utilized (Chang, 2001). Hence, through the use of the right software metrics, software developers are enabled to access the quality before building their products. In that way development teams understand, control and improve what they do and the way they do it (Fenton et al. 1998). As a result, development teams are able to understand where the product being developed stands in terms of maintainability, complexity, size and cohesion. High quality products are likely to increase efficiency and profitability while they decrease the manufacturing cost in the long run (Garvin, 1984).

In essence, Pulford et al. (1995) summarize in their book the motivations for using metrics which are stated below:

- Project planning, estimation and tracking
- Perceive quality and business objectives
- Improved software development communication

2.6.3 Software quality metrics usage challenges

No matter the development process, measurements through metrics assist the programmers to inspect their code and make the appropriate improvements needed during the construction face. By effectively using them, software failures can be prevented, thus problems can be identified before they worsen (Ewusi-Mensah, 1997).

However, although their highlighted importance and the greater control metrics can offer to the software development process, they seem to be complex and difficult to use for many companies (Gopal, 2002) while further research indicates that two in three metrics implementations fail (Pfleeger, 2008). During a SDLC, managers, developers and testers are involved and make use of metrics. According to Orlikowski et al. (1994) when different groups of people are using a technology leads to varying perspectives. The result might be communication problems during the development and hence resistance to use software quality metrics, leading to less-controlled and riskier software development process. Huisman and Livari (2006), argue that managers perceive the productivity and the quality of the development process to be more important, while developers perceived the methodology support and the validation more crucial. Similar to this study, Sheetz et al. (2009) found that managers perceive software quality metrics as more useful than developers. Hall and Fenton (1997) compared metric programs and reported that managers were more enthusiastic with the use of metrics, while developers lacked motivation to collect metrics data whose accuracy was disbelieved. Consequently, in order to adopt and effectively use software quality metrics both managers and developers should share the same perspective regarding with the usage of the measurements. It would be crucial for managers to understand which metrics development teams prefer to use in order to develop the right strategies for encouraging the appropriate use. If those two parties disagree about how and which metrics should be used then these may be used incorrectly resulting in a less effective system development process (Orlikowksi et al. 1994).

Moreover, the measurements provided by the metrics are not the goal. The main goal is to analyze and make improvements through the feedback derived from the metrics. Some examples may include increase in software productivity and reliability and project planning improvement.

However, there are multiple companies which are putting the pressure to the development teams in order to meet strict deadlines and produce even more features in in the production (Elgebeely, 2013) resulting in development teams who consider the whole development work to be an overriding priority. Hence, the use of internal measurements may cause extra delays at the implementation phase of a software while developers find them problematic. Umarji et al. (2009) conclude in their research that learning to use metric tools had been tedious while developers spent too much time reporting the measures derived from these tools. This fact made them think of metrics as an overhead. Similar to Umarji et al. (2009), Czewronka et al. (2015) highlight that the focus on software quality feedback slows down the development process. Due to these cases, the use of metrics takes the form of extra work and as a result they do not help the organizations to achieve their objectives. Thus, a lot of metric programs fail because of the high overhead nature of the metrics and the burden placed on the developers' side during the SDLC (Dutta, 2009). Last but not least, Fenton et al. (1998) argue in their research that software quality metrics can be used in order to monitor developers' productivity.

2.7 Traditional software development process metrics

In chapter 2.1 we defined the SDLC and stated the most well-known models of software development processes. During the two upcoming subchapters we will focus on the traditional and the agile software development process.

As far as the Traditional Software Development (TSD) process is concerned, according to Kan (2002) software metrics are classified into product, process and project metrics. Rawat (2012) further explores these types of software metrics and he assigned the following characteristics:

1.Product metrics: They measure the size of the program, complexity, performance, portability, maintainability and product scale. Product metrics are used to measure the medium or the final product. Product metrics describe the characteristics of a product such as:

- Size
- Complexity
- Portability
- Reliability

2. Process metrics: These metrics aim at process duration, cost incurred and type of methodology used. Process metrics can be used to increase software development and maintenance. Some examples:

- Time needed in order to produce a product
- Effort required in a process
- Defects found during the testing process

3. Project metrics: The metrics of this category are used to monitor the project status and thus help to optimize the software development plan. Some examples:

- Productivity measurement
- Number of developers
- Cost measurements and schedule

Scotto et al. (2006) further categorize the product metrics into two categories:

- Static metrics: These types of measurements are useful in order to understand the maintainability, complexity and understandability of the systems and are based on system representations.
- Dynamic metrics: These metrics are collected during the system execution and thus can be useful in order to assess the efficiency and the reliability of the system.

In the TDS software complexity is measured in order to reduce the complexity of the code and further reduce the maintenance costs. In this development method the metrics used in order to measure software complexity are the Lines of Code (LOC), Halstead Complexity Metric (HCM) and the Cyclomatic Complexity Metric (CCM). Furthermore, Rawat et al. (2012), presented the Source line of code (SLOC) metric, the Object-oriented metrics and the Function point metrics. Database metrics and duplicate metrics - in order to measure the duplicate code- were discussed by Rentrop (2006).

2.8 Agile software development process metrics

The fast-technological changes taking place require that organizations adapt to an agile environment that is constantly changing (Kassim et al., 2004). Consequently, even more organizations are adapting the Agile way of working in order to react to the fast-paced changing environment.

There are studies indicating the increasing interest in Agile Software Development method and in agile metrics (Dyba et al.,2008). This increase for the Agile software development method rose the need for metrics. However, the metrics used in the TSD could not be directly used in an Agile environment (Kunz et al., 2008). Oza and Korkala (2012) in their research presented a graph of metrics that are relevant to the Agile development method used in different areas more frequently. Among the categories there are metrics regarding testing, code, automation and engineer.

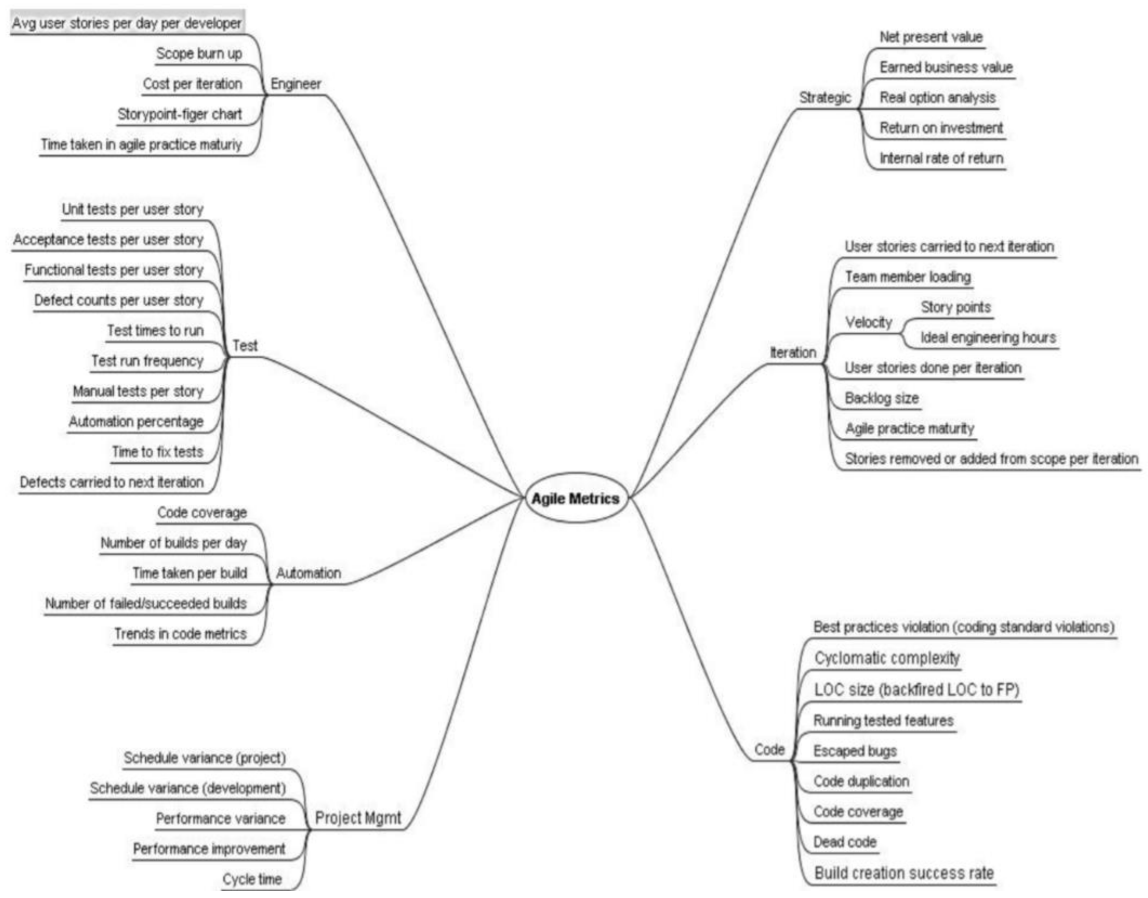


Figure 5: Agile Metrics (Oza & Korkala, 2012)

The same research classifies the metrics in three different categories:

- Code level metrics: These metrics provide visibility into the quality of the code
- Productivity: Support decision making by providing burn-down charts and project size units
- Economic: Similar to productivity metrics, these metrics provide support to the decision-making processes.

In essence, software metrics in an Agile environment may optimize the work and in such an agile way of working, these metrics should be simple and easy to maintain (Downey et al.,2013).

2.9 Literature - software quality metrics

As mentioned in the previous chapters, software quality metrics play a crucial role during the SDLC. In this section we will review the literature regarding the software quality metrics. There will be a review of research papers and the overview of the metrics discussed can be found in Appendix A.

Survey on Impact of Software Metrics on Software Quality

Rawat et al. (2012) performed a case study related to the software quality of Boeing 777 project. The project was a huge leap toward software quality with a million lines of code while it was critical to ensure the best software quality practices and implementation. However, each vendor of the project was using different metrics resulting in a snowball situation while it was hard to understand the progress of the project. Luckily Boeing realized in an early phase the importance of using the right metrics in order to pursue software quality and identify the possible risks early, correct them and avoid any delays in the project. According to Rawat et al. (2012), this also led to the following advantages:

- The use of right metrics allowed each project to flow smoothly without any roadblocks.
- The communication between Boeing and its vendors was smooth while they were both sharing the same metrics.
- Constant monitoring through metrics around coding and testing made sure that the project was a success.

A number of metrics was proposed and exercised in order to measure the quality of the system before the implementation of this project. These metrics are shown in the table below.

Metric Name	Definition
SLOC	Source lines of code metrics
FP	Function Point metrics
OO	Object-Oriented metrics

Table 3: Software Metrics (Rawat et al. 2012)

Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development

This study conducted by Chawla & Kaur (2013) addresses the importance of software measurement on behalf of the complexity of the current systems. According to the study software quality metrics are used for :

- Quality planning
- Process improvement
- Quality control
- Reliability estimation
- Analysis of customer satisfaction
- Reduce the software maintenance costs

Chawla & Kaur (2013) argue that measures and complexity alone are not enough in order to provide accuracy in maintaining the systems so they suggest the use of object-oriented metrics. The study concludes that metrics help in order to determine the complexity and maintainability of the code. The metrics proposed by Chawla & Kaur are stated below:

Static Code Metrics	**Definition**
SLOC	Source Lines Of Code
CP	Comment Percentage
HM	Halstead Metrics for complexity measurement
CC	Cyclomatic Complexity
Object-Oriented Metrics	
WMC	Weighted Method Per Class
DIT	Depth of Inheritance Tree
NOC	Number of Children
CBO	Coupling Between Object Class
RFC	Response of a class
LCOM	Lack of Cohesion
MOOD	Metrics For Object Oriented Design

Table 4: Software Metrics (Chawla & Kaur, 2013)

Metrics and models in software quality engineering.

Kan (2002) in his book presents an overview of the most important software quality metrics. He classifies software metrics into product, process and project. Kan (2002) argues that software quality metrics are a subset of software metrics and are mostly associated with the process and product metrics. Software quality metrics are then further divided into end-product, in-process and maintainability metrics.

Product Quality Metrics	**Description**
DD	Defect Density(Loc, FP)
MTTF	Mean Time to Failure (time)
Customer problems	Measures the customers' problems
Customer satisfaction	Use of surveys based on a five-point scale
In-Process Quality Metrics	
Defect Density	Testing after code is integrated into the system library
Defect Arrival	Measures the defects arrivals before testing and releasing the software to the field
Phase-Based Defect Removal Pattern	Extension of the test Defect Density. Tracks the defects at all phases of the development process.
Defects Removal Effectiveness	Measures the team's ability to remove defects before release.
Software Maintenance Metrics	
Fix Backlog	Workload statement of reported problems that still remain at the end of each week.
Backlog Management Index	Ratio number of closed or solved problems of each month.
Fix Response Time & Responsiveness	Measures the mean time for all problems from open to closed. Related with customer satisfaction.

Percent Delinquent Fixes	Measures the number of fixes that exceeded the response time against the ones that delivered on time and names them as delinquent.
Fix Quality	Measures the percent of the fixes that are defective (could not be fixed).

Table 5: Software Metrics (Kan, 2002)

Towards a Catalog of Object-Oriented Software Maintainability Metrics

Due to the high usage of Object-Oriented programming and the importance of software maintenance, Saraiva et al. (2013) categorized a range of metrics derived from a large number of researches. Thus, they gathered a high number of researches that studied software maintainability and summarized them in a catalogue providing information about metrics. In that way Saraiva et al. (2013) are aiming to help researchers and practitioners in the identification of the appropriate metrics adoption. The domain categorization proposed by the paper, presents the following metrics categories: Evaluated metrics, industrial, academic, open source, internal attribute association, external attribute association, internal, external, most adopted, most relevant, isolated adopted, duplicated, correspondent and tool aided. Furthermore, the metrics presented in this study are the following:

Metric Name	Metric Description
NASSOC	The total number of associations
NAGG	The total number of aggregation relationships within a class diagram
NAGGH	The total number of aggregation hierarchies within a class diagram
NGENH	The total number of generalization hierarchies within a class diagram
WMC	Weighted Methods per Class
DIT	Depth of Inheritance
NOC	Number of Children
RFC	Response for Class
CBO	Coupling Between Objects

LCOM	Lack of Cohesion Methods
NC	Number of Classes
NMC	Number of Methods
CBO	Coupling Between Object Classes
NOC	Number of Children
LOC	Lines of Code
CCN	Cyclomatic Complexity Number
MPC	Message Passing Coupling
WAC	Weighted Attributes per Class
NoM	Number of Methods
DC	Degree of Cohesion
CTA	Coupling Through Abstract Data
CR	Comment Ration

Table 6: Software Metrics (Saraiva et al., 2013)

Empirical Studies on Quality in Agile Practices: A Systematic Literature Review.

This study addresses the issues of software process and product quality in agile methods through two ISO standards, the ISO/IEC 12207 and ISO/IEC 9126. They argue that Agile development redefines quality assurance. Through the use of practices such as pair programming and test-driven development, software delivery is delivered faster, with a higher acceptance by the users and of higher quality (Sfetsos & Stamelos, 2010). Furthermore, according to them through pair programming developers serve a continual design and code reviews which results in less defects and higher code quality. Moreover, through test-driven development developers use test cases in a detailed design with refactoring in order to make small changes without changing system's external behavior. Sfetsos & Stamelos (2010), studied 46 researches.

According to the paper, internal quality is usually measured by different metrics (similar to the ones mentioned at the previous researches) such as: code size, cyclomatic complexity, coupling and cohesion. In addition to that and according to the study, pair programming has increased products' quality and played a significant role in areas such as:

- Code quality
- Better teamwork and communication within the teams
- Better code understanding
- Information transfer

The research concludes that agile methods such as extreme programming and scrum leads to better work estimation while the use of refactoring increases quality leading to higher customer satisfaction.

Software Metrics for Agile software Development

This paper addresses the refactoring method in an Agile software development process by the use of measurements in order to achieve high quality at the end product. Refactoring -the change of source-code and software design without altering the external behavior- has an essential importance and hence there is need for a new set of metrics. It is also highlighted that the cost of change in an Agile software development is less than the traditional methods and this is achieved through iterations. The metrics proposed from Kunz et al. (2008) in this study are the following:

Metric Name	Metric Description
NNP	Number of Name-Parts
NC	Number of Characters
CL	Number of Comment-Lines
NLV	Number of Local Variables
NCO	Number of Created Objects
NRO	Number of Referring Objects
NP	Number of Parameters
LOC	Lines of Code
CBO	Coupling Between Objects
NOC	Number of Children
DIT	Depth of Inheritance Tree
CCDG	Cycle Count of Dependency Graph

Table 7: Software Metrics (Kunz et al., 2008)

The study is classifying the metrics in three categories:

- Method-Level
- Class-Level
- Package-Level

Kunz et. al (2008), conclude that through refactoring in an Agile development process, the early observation of quality through metrics is essential. Furthermore, they argue that the metrics should be simple and flexible in order to support the development and impact the software's quality.

3. Research Methodology

The upcoming chapter aims to present the research approach and the research design followed during this study. Hence, the following chapters explain the research approach used, the choice of the literature review and the interview analysis.

3.1 Research approach

The research approach we will give a structure to our research. In order to do that, a literature review has been conducted in Chapter 2. Through the literature review, we tried to interpret the available research relevant to the topic area (Kitchenham, 2006) and give an understanding to the reader. Furthermore, a combination of surveys and semi-structured interviews with various experts, as well as the use of secondary data will be used in order to identify the challenges in practice. Through the surveys/interviews and the secondary data the researcher will have the opportunity to find out what concepts and terminology experts use. The questions will be pre-determined while the respondents will be allowed to answer in their own way making it possible for the interviewer to derive more information in promising areas and extract real-life experiences from target audience that makes use of quality metrics. Hence, a qualitative research approach will be followed. In figure 6, a representation of the research approach is shown.

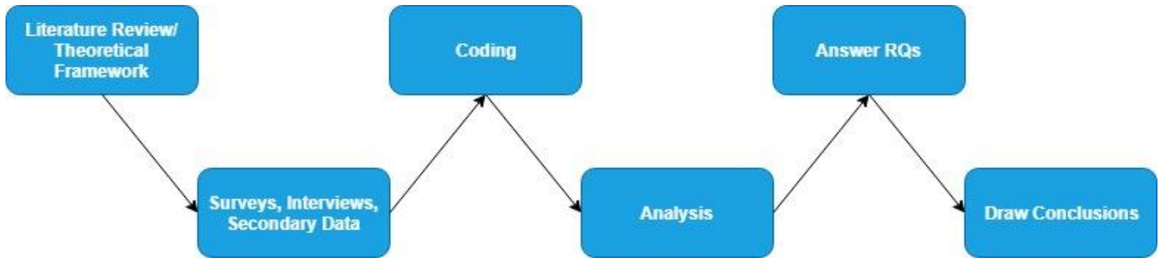

Literature Review/ Theoretical Framework
Surveys, Interviews, Secondary Data
Coding
Analysis
Answer RQs
Draw Conclusions

Figure 6: Research Approach

.

Consequently, the research is based on data collection from experts involved with software delivery. However, the exploration of the literature review took place first in order to gain the appropriate understanding of the research topic. Subsequently, the review of the literature review will help the researcher to define both the survey and the interview questions. After the construction of the questions, we will communicate with resource people in order to get the insights for this research and shed light to the topic of software quality metrics while identifying the challenges in practice.

3.2. Literature review

As previously mentioned, the purpose of the literature review was to gain the appropriate understanding of the research topic. Hence, the first step before the selection was to clarify what literature was useful in order to create the theoretical framework.

The literature was derived from papers, articles and books from online databases such as Google Scholar, the University Library Catalogue, Academia.edu and IEEE. The most common keywords used were: Software quality, Software metrics, Agile metrics, Software quality model, Software development metrics, etc. These keywords were used in different order as well in order to come up with the most relevant literature regarding the research. The papers were selected after reading the abstract, introduction and the conclusion. After the collection of multiple papers, the literature review started

3.3 Data collection

Based on the literature results, the next step was to define a survey and an interview questionnaire based on the challenges and needs of the use of software quality metrics. The scope of the survey and the interviews included various developers. Consequently, the outcomes are valuable in order to further investigate the challenges and their perspective regarding software quality metrics and identify ways in order to successfully integrate them in the development process.

From each organization the interviewee was asked to fill in the survey and then he/she was interviewed. The survey consisted of 16 questions of three forms. The questions included multiple choice questions, open questions and questions based on Likert's scale. The software used was Qualtrics.

The interviewees were asked upfront whether they had a problem being recorded during the interview. Recordings were used in order to accurately transcribe each interview and avoid possible mistakes. Moreover, through the semi-structured interviews the researcher aims to gain a consistent line of inquiry while avoiding being too rigid. Last but not least there were secondary data collected from

interviews that took place with people involved into the software delivery. In total 13 interviews took place by the researcher, while the secondary data consisted of 37 interviews. Hence, in total we analyzed 50 interviews.
In the table below we present the questions that formulated either the survey or the interviews and we relate them to the research questions stated in Chapter 1.

Question Number	Interview/Survey	Question/Statement
1	Interview	What is your role and responsibility within the company?
2	Interview	How many years of software development experience do you have?
3	Interview	What is your current assignment and from how many people does your team consists of?
Research Question		**How can software quality metrics be integrated in the development process in an unobtrusive manner?**
4	Survey	There is adequate time in my project for metrics tasks.
5	Survey	My company provides the necessary training in order to use software quality metric tools.
6	Survey	People knowledgeable about metrics, are easy to approach within my company.
7	Survey	I receive guidelines for usage of the software quality metric tools I use.
8	Survey	Using software quality metric tools and reporting data from them takes too much time
9	Survey	Learning how to use software quality metric tools was a tedious task.
10	Survey	Software quality metric tools are easy to use.
11	Survey	Too much mental effort is needed in order to report data derived from the software quality metric tools.

12	Survey	The data generated by the metrics are actionable.
13	Survey	My team members are enthusiastic about software quality metrics.
14	Survey	I am satisfied with the effectiveness/efficiency of the software quality metric tools I use
15	Survey	The software quality metric tools used are well-integrated with the other tools used.
16	Survey	Using software quality metrics during my projects is: Satisfying, Annoying, Worthless, Valuable, Purposeful, Purposeless, Enjoyable, Unenjoyable.
17	Survey	I feel that the use of software quality metrics can be used in order to monitor my performance
18	Survey	I agree with employers using software quality metrics in order to evaluate employee performance.
19	Interview	How much time do you spend on reviewing software quality metrics (percentage)? Can you elaborate?
20	Interview	Is there time available to fix complexity or duplication issues in your code? What is its percentage over your total working time?
21	Interview	What are the obstacles in your opinion for the quality metrics to be easily integrated?
22	Interview	What kind of improvements would you expect in these tools?
Research Sub-Question		**What are found to be the most effective software quality metrics?**
23	Survey	What software quality metrics do you currently collect from the ones listed below? Please mention any metrics used that are

		not in the list. (Lines of Code, Defect Density, Lack of Cohesion, Cyclomatic Complexity, Code Duplication, Weighted Method per Class)
24	Interview	What are the most effective software quality metrics that you use? Why these?
25	Interview	When do you deem a quality metric as an effective one?
Research Sub-Question		**How do organizations measure their software quality?**
26	Interview	How would you describe the quality of your software? How do you measure it
27	Interview	What methods or tools does your team use in order to improve software quality and why?
28	Interview	How much has the use of metrics improved your projects' performance?
Research Sub-Question		**How does the development process impact the software quality metrics integration?**
29	Interview	What type of development method do you use?
30	Interview	Can the development method have an impact on software quality metrics integration? Can you elaborate?
31	Interview	Is there any other information that you would like to share with me?
32	Interview	Are you available for follow- questions?

Table 8: Interview/Survey Questions

3.4 Data Analysis

After the data were collected the following steps were used in order to analyze them:

1. Organize the data. During this step all the data were organized into 2 categories. The first category included the data collected from the interviews in chronological order. The second category included the secondary data provided.
2. Data review. This step included a review of the material collected during the data collection.
3. Coding. The data collected both from the interviews and from the secondary data were mapped in an Excel file. The vertical axis had labels like “Metrics used”, “Obstacles”, “Improvements”, “Time reviewing metrics” etc. A sample of the document can be found in the Appendix 8.4 of the research.
4. Interpretation. This step involved the visualization of the findings by linking the results to the research questions. This step can be found in the next Chapter, where the results of this research are shaped.

4. Results

This chapter presents the results derived from the surveys, the interviews and the secondary data with the resource people. The results are presented either by text or by graphical representations.

The first section of the chapter provides an overview of the people involved with the data collection while in the second section the data from the surveys and the interviews are analyzed.

4.1 Cases overview

In this section we will provide an overview of the interviewees, their roles and their experience. The interviewees that took part in this research work in organizations of different industries. However, all of the interviewees were involved into software delivery. Their names were kept confidential.

Interviewee	Interviewee Role	Interviewee Experience
A	Software Engineer	25 years
B	Junior Software Developer	1 year
C	Senior Software Developer	8 years

D	Software Developer	5 years
E	Lead Developer	5 years
F	Software Developer/Head of IT	5 years
G	Senior Software Engineer	7 years
H	Lead Software Developer	15 years
I	Software Engineer	2.5 years
J	Software Developer	13 years
K	Software Developer	5.5 years
L	Software Developer	5 years

Table 9: Interviewees characteristics

4.2 Software quality in organizations

Offering high quality of software is essential for the organizations today in order to satisfy the needs of their customers. Hence, software quality plays a significant part of attention while its importance is steadily increasing. In the following sections we will present the data collected regarding the research question “How do organizations measure their software quality?

In order to shape the answer to this research question we will analyze 3 questions that were asked during the interviews with the 13 resource persons. These questions are stated below:

1. How would you describe the quality of your software? How do you measure it? (Subchapter 4.2.1)
2. What methods or tools does your team use in order to improve software quality and why? (Subchapter 4.2.2)
3. How much has the use of metrics improved your projects’ performance? (Subchapter 4.2.3)

Moreover, we will also make use of some data relevant to the question derived from the secondary data.

4.2.1 Software quality description and measurements

After the introducing questions during the interviews all of the 13 interviewees were asked about the quality of their software and how do they measure it. The results are displayed in the following table.

Interviewee	How would you describe the quality of your software? How do you measure it?
A	Very good quality. We use measurements from tools. Awareness is increasing
B	We plan tests and use tools for code maintainability
C	We measure quality through customer satisfaction
D	We do not measure quality. Measuring quality is important but also a slowdown in the overall delivery
E	We let other people judge it. Use of static analysis tools, metrics
F	For some projects there is no time. Use of unit tests, user interface and more stats through the continuous integration
G	Really high. We measure it through code coverage, by different levels of testing, metrics and tools provided by our company
H	Quality is improving because we are rewriting old code. We use story points and measure velocity
I	We try to have less bugs, readable code with improved functionality and happy customers
J	We use metrics in order to make sure that our application works
K	We use various metrics through tools and unit tests. We have a code check once a year by an external company
L	We do tests for al the implementations. The performance should be better or the same as the last release
M	We are changing to that direction by using tools to analyze our code

Table 10: Software quality description and methods

From the secondary data we derived the following information/statements:

- Deadlines are more important than quality
- Companies are materialistic. They prefer fast and bad code than long and good
- Businesses want more results than quality. Delivering working code is a higher priority than adding measurements
- We do not measure quality but we do tests to improve it
- We use ratings and analytics to improve quality
- We experience the disadvantages of not using metrics
- These tools are required for every project
- The focus on releasing fast is bigger than walking through the code
- Quality is subjective
- We test our software; metrics are support tools
- Quality is easy to measure but hard to explain. It is shown by the number of bugs
- If we meet our customers' requirements, we have a good quality
- Software quality metrics tools increase software quality

It is remarkable to mention that the part of the question that says "How would you describe the quality of your software" was barely answered by our 13 interviewees. However, through the answers of these questions and the secondary data, we gain a broad understanding of what do the experts think of quality and how they approach it. Software quality conceptualization seems to be a vague concept for our participants. Moreover, we collected multiple ways that our resource people use in order to measure software quality.

4.2.2 Methods and tools used in order to improve software quality

Each company follows different methods in order to attain higher software quality. In the next graph we will display the methods that were reported both during our surveys/interviews and by the secondary data.

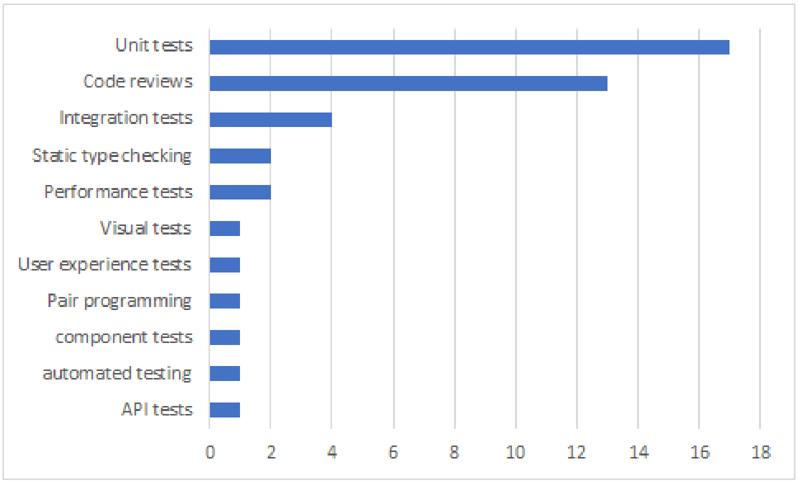


Figure 9: Methods used to improve software quality

According to Figure 9, unit tests appeared 17 times in our data as a method in order to attain higher quality of software. Furthermore, code reviews comes second which appeared 13 times. Last but not least, integration tests (4) , static type checking (2) and performance tests (2) were the methods that were mostly derived from our data regarding software quality.

Similar to the previous figure, in order to analyze the tools used to improve software quality we will analyze the data derived both from the 13 surveys/interviews and from the secondary data. We will depict the tools in alphabetical order mentioning the number of their occurrence within our data in a parenthesis next to their names. Furthermore, we will provide a small description which is derived from the tools' official website. The following table displays the tools used as mentioned by our interviewees.

Tool (No of occurrence)	Description
Apache Subversion (1)	Acts as a control system for tracking changes to files, folders and directories
Bitbucket (1)	Control respiratory repository hosting service
Checkstyle (1)	Static code analysis tool
Code Climate (1)	Static code analysis tool
CodeShip (1)	CI/CD platform
Confluence (2)	Collaboration tool
Crucible (1)	Collaboration tool
ESLint (1)	Linter tool for identifying and reporting on patterns in JavaScript
Git (4)	Distributed version-control system for tracking changes in source code during software development.
Github (4)	A web-based hosting service for version control using Git
Gitlab (2)	A web-based DevOps lifecycle tool that provides a Git-repository manager providing issue-tracking and CI/CD pipeline features
Grafana (1)	Software for analytics and monitoring

IntelliJ IDEA (2)	A Java integrated development environment - static analysis
Jenkins (2)	Automation server for building, deploying and automating projects
Jira (10)	A tool for bug tracking, issue tracking and project management
JSLint (2)	A static code analysis tool for JavaScript
Junit (1)	A framework for writing and running automated tests
Kibana (1)	Software for analytics and monitoring
Laravel (3)	PHP web framework for web-app development
Lint (2)	Analyzes source code to programming errors, bugs, stylistic errors, suspicious constructs
MagicDraw (2)	A UML, SysML, BPMN, UPDM modeling tool
MyPy (1)	Static type checker for Python
Net Promoter (1)	Management tool for customer satisfaction
New Relic (2)	Performance monitoring tool
Prometheus (2)	Records real-time metrics
Protactor (1)	Test framework for Angular and AngularJS apps
PyCharm (2)	Integrated development environment for Python
Pylint (1)	Bug and quality checker for Python
Redmine (1)	Open source project management and issue tracking tool
Reshaper (1)	Code quality analysis, error tracking
Selenium (1)	Builds UI web tests
Sentry (1)	Application monitoring, error reporting
SonarQube (9)	Inspection of code quality
Stackdriver (1)	Analytics and monitoring
StyleCop (1)	C# source code analysis

Symphony (1)	Collaboration tool
TeamStudio (1)	Performance monitoring tool
Travis (1)	CI service for building and testing projects hosted at GitHub
TSLint (1)	Static analysis tool that checks TypeScript code for maintainability and functionality errors

Table 11: Tools used in order to improve software quality

Table 12 depicts all the tools as mentioned by the resource persons interviewed. Not all of these tools offer metrics. However, they were mentioned as a means to attain higher software quality. It is worth mentioning that Jira (10) and SonarQube (9) were the most referred tools. Both of these tools provide insights (metrics) regarding software quality.

4.2.3 Use of metrics and improved performance

All of our 13 interviewees were asked whether the use of metrics improved their projects' performance. The answers are displayed in the following table.

Interviewee	How much has the use of metrics improved your project's performance?
A	Metrics are useful. They help you to make the code more maintainable
B	Metrics improve your understanding on how every program is being developed
C	Not that much. It is nice that you can immediately see the warnings and fix them
D	N/A
E	The insights we are getting from metrics improves the outcome
F	N/A
G	The use of metrics played an important role. It is crucial to have everything tracked and automated
H	Metrics makes us go faster

I	By sticking with metrics, you improve performance in the long run and the maintainability of your product
J	A lot of metrics helps a lot but some others do not. They allow developers to be aligned with the best practices.
K	Metrics improved our code quality comparing to the past
L	Software quality metrics tools make a difference
M	We are in the early stages, so it is hard to say yet.

Table 12: Software quality metrics - Performance improvement

From the secondary data we derived the following information:

- Software quality increases working with these tools
- Through software quality metric tools you can see if a software is easily maintained
- Using these tools will improve the quality of the code

Hence, and according to the results there were interviewees that do not use software quality metrics and did not reply. Furthermore, for most of our respondents the use of software quality metrics leads to a more maintainable product. Last but not least, 3 of our respondents A, H and K made a comparison of how poor-quality checking was in the past and how it is now with the use of these tools. It is remarkable to mention that these respondents have 25, 15 and 5.5 years of experience in the field.

4.3 Effective quality metrics

Software quality metrics are the source of information through which people involved with software delivery can make decisions regarding the software developed. Through them, the experts are able to measure and predict the necessary software resources for a project. However, it is vague when it comes to the effectiveness of these tools and measurements. In the following section we will analyze the software quality metrics so that to answer the following research question: “What are found to be the most effective software quality metrics? “ .

In order to shed light to this question we will analyze the answers of 3 questions that were asked both at the survey and during the interviews. These questions are stated below:

1. What metrics do you currently collect from the ones listed below? Please mention any software quality metrics that you use that are not on the list. (Subchapter 4.3.1)

2. What are the most effective software quality metrics that you use? (Subchapter 4.3.1)
3. When do you deem a software quality metrics as an effective one? (4.3.2)

4.3.1 Collected quality metrics

In this section we will display all the metrics that were collected. We will include data derived both from our surveys/interviews and from the secondary data gathered. The following Figure represents the data gathered.

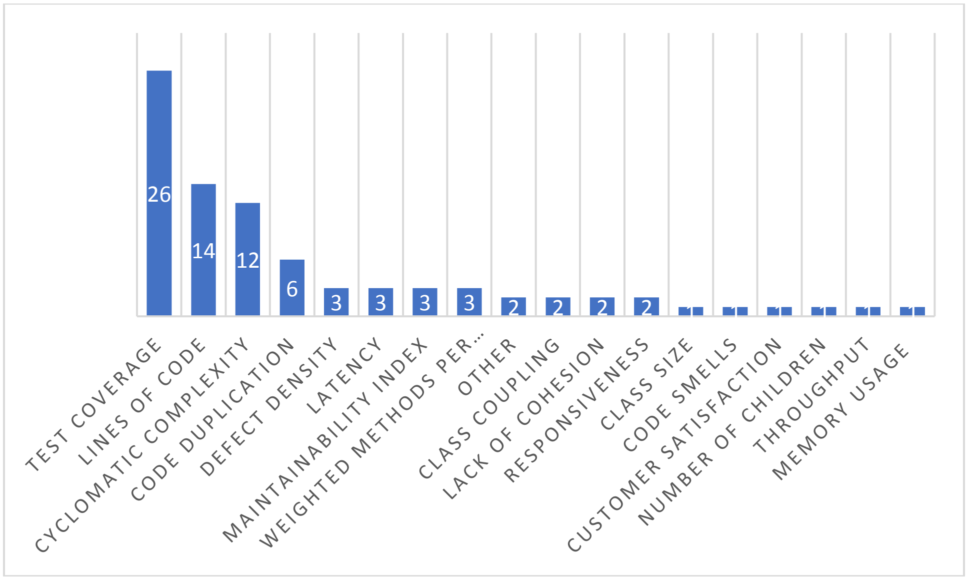


Figure 8: Quality metrics used

Figure 8, represents the occurrence of the quality metrics. The vertical axis displays the number of the resource people that use the metrics that are displayed in the horizontal axis.

Based on the visualization of Figure 8 there are some metrics which appear more frequently than the others. These metrics are:

1. Test Coverage (26 appearances)
2. Lines of code (14 appearances)
3. Cyclomatic complexity (12 appearances)
4. Code duplication (6 appearances)

4.3.2 Effective software quality metrics

All of the 13 interviewees were asked when do they deem a software quality metric as an effective one. The answers to this question are visualized in the table below.

The responses are based on the participants' experience regarding software quality metric usage.

Interviewee	When would you deem a software quality metric as an effective one?
A	They can make the code more maintainable, modular and well designed
B	They can be read by the business people
C	Fast metrics
D	N/A
E	When they provide more guidance to the user
F	Big memory, latency, detect errors, can cover the whole system
G	Result in a good program and the customer is satisfied
H	Provides lots of alerts, warnings, errors
I	Gets the results on time and maintain the quality over time
J	When you see the impact on performance and bugs
K	Results in understandable and maintainable code
L	Results in maintainable, secure and fast product
M	When you have reasonable amount of bugs and issues reported

Table 13: When a software quality metric is effective

Some of the characteristics mentioned in Table 10 are mentioned more than once from the interviewees. Hence, we could say that a software quality metric is effective when It provides fast insights regarding bugs and results in a maintainable product.

4.4 Software quality metrics - development process

There are different models of software development processes. Each model provides the team with different phases in order to analyze, design, program, test, install and maintain the end product. However, it is not clear whether the development process might impact the software quality metrics integration. In order to answer this research question, we will analyze the following questions:

1. What type of development method do you use? (Subchapter 4.4.1)
2. Can the development method have an impact on software quality metrics integration? Subchapter (4.4.2)

4.4.1 Development methods followed

The following bar chart displays the development methods that were mostly used by the resource people. The figure contains data from 50 people, gathered both from the interviews and from the secondary data.

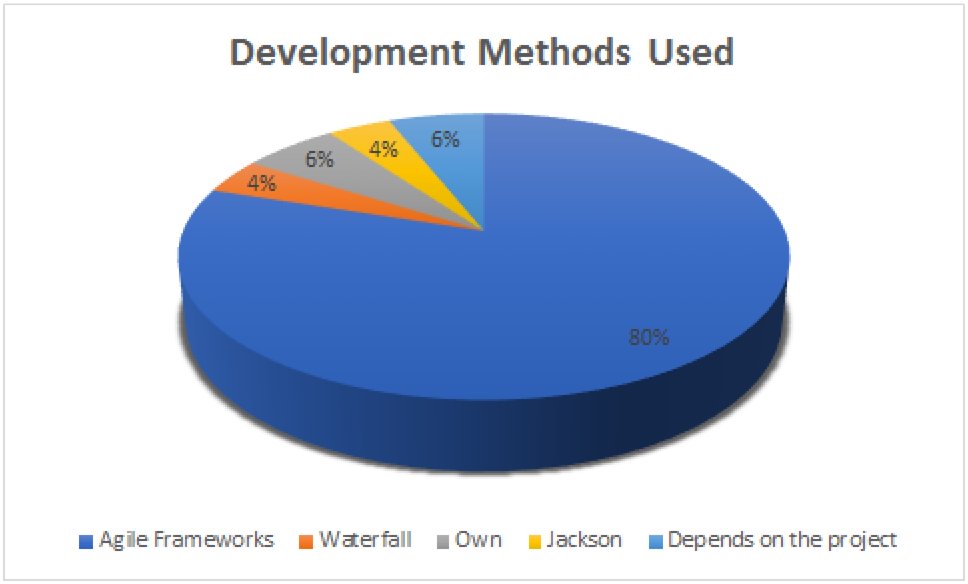


Figure 9: Software Development Methods used

Based on the results above, we acknowledge that Agile is the most widely used methodology since it appeared 40 times. Furthermore, Agile frameworks appear in many situations as multiple teams use some of its principles in combination with other methodologies. It is remarkable to mention that there where interviewees with more than 10 years of experience who used to work with different methodologies in the past but changed to Agile.

4.4.2 Impact of development method in metrics integration

In the following table, the responses regarding the question" Can the development method have an impact on software quality metrics integration?" are displayed using the data derived from the 13 interviews.

Interviewee	Can the development method have an impact on software quality metrics integration?
A	Metrics can be easier integrated through Agile - there is too much documentation with Waterfall
B	Easier with Scrum
C	It impacts - Kanban works better
D	It depends on the requirements of the project
E	Depends on the type of metrics
F	The development method impacts everything
G	The Test-driven-development is the most important factor that impacts metrics
H	Yes, it is easier with Scrum because of the plan-do-check-act, there is continuous improvement so the measurement tools are useful
I	Yes, also the quality of the sprints and the project manager
J	In Agile yes because you release fast, so you test a lot to improve the features for the next sprints
K	No experience with other methodologies to answer
L	Obviously. In Scrum you have to be aware every day
M	Depends both on the development method and the organization

Table 14: Development process - Software quality metrics integration

As mentioned in the previous subchapter most of the resource persons operate in their teams with the Agile methodology and its frameworks. There were some references regarding the connection of good quality product and Agile methodology. Based on the results above, we acknowledge that there are some answers that appear the most and can shape a concluding remark. Hence, for most of our interviewees the development method impacts the software quality metrics integration. For some of them through Agile - and due to its iterating processes- teams seek for a continuous improvement that results in more tests. As a result, we can conclude that software quality metrics can be easily integrated in an Agile environment.

4.5 Software quality metrics integration in the development process

In order to answer the main research question, we will mostly focus on specific questions addressed both in the survey and during the interviews to our 13 interviewees. Moreover, there will also be information collected from the secondary data. Sections 4.5.1- 4.5.3 address the data collected during the interviews and some relevant information found in the secondary data. At section 4.5.4 we will display part of the data relevant to the questions derived from the survey with our 13 interviewees.

4.5.1 Integration obstacles

For the purpose of getting an understanding of the difficulties, we addressed the following question to the 13 of our interviewees:
What are the obstacles in your opinion for the quality metrics to be easily integrated?

The data derived from this question are depicted in the following chart:

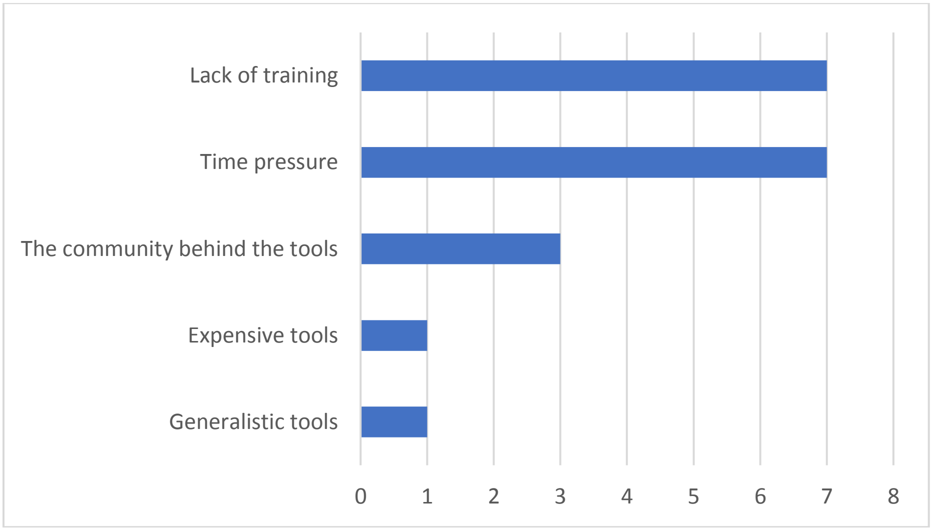


Figure 10: Integration Obstacles

Hence, our respondents highlighted that time pressure and lack of training are the two most significant barriers that affect the software quality metric integration. Moreover, the poor community behind the tools used was also mentioned as an obstacle meaning that there is not proper support in order to guide the users as far as the outputs translation is concerned. Last but not least a respondent stated that the tools used, are composed by hard coded rules that do not apply to every development team while some of these tools are expensive.

4.5.2 Areas of improvement

Subsequently, after gathering information regarding the obstacles of the software quality metrics, we tried to capture the thoughts of our resource persons concerning the improvements that they would expect from these tools. We managed to attain that by asking our interviewees the following question:
What kind of improvements would you expect in these tools?

The results are displayed in the following chart:

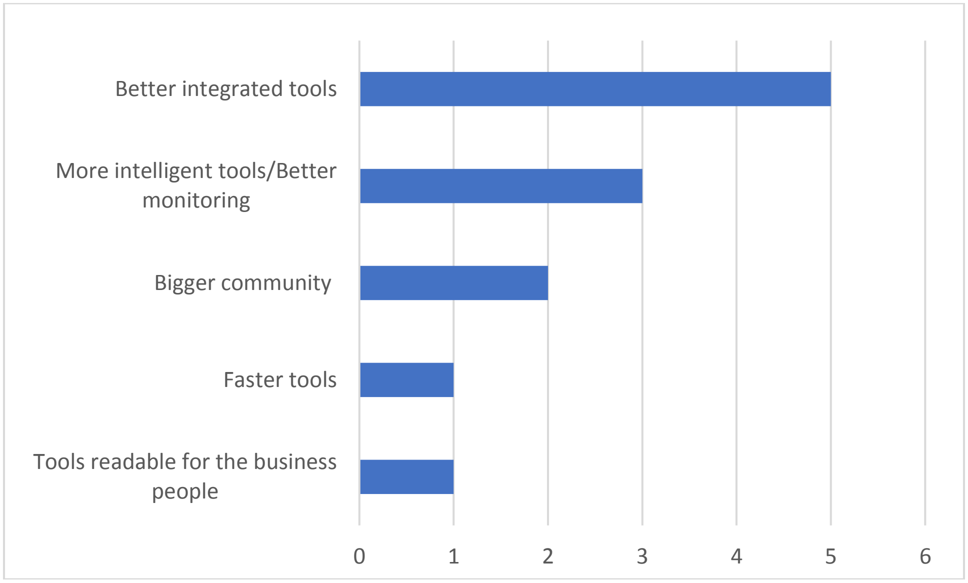


Figure 11: Improvements areas

According to the data of the previous chart, there were many different opinions regarding the expected improvements of these tools. However, the one with the highest occurrence is that these tools should be better integrated with their coding environments providing fast and accurate results to the developers. It is important to mention that there should be more guidance behind these tools (also mentioned in the obstacles section) while experts should become more knowledgeable about them.

4.5.3 Time spend reviewing metrics

Interviewee	How much time do you spend on reviewing software quality metrics (percentage)?
A	10-20%
B	20%

C	N/A
D	N/A
E	10%
F	10-20%
G	0
H	10-20%
I	15-20%
J	10%
K	2%
L	30%
M	10-20%

Table 15: Time spent reviewing metrics

According to the results of this question we could say that from the interviewees that use metrics, most of them spend 10-20% of their time reviewing them. Interviewee B finds that a small percentage, however time pressure was behind that answer. Interviewee E spends less than 10% due to the fact that their code is new and they mostly advice metrics for the problematic areas. Moreover, interviewee F stated that due to the fact that they work in an Agile environment they are tracking metrics daily, however this amount is 10-20% for developers. An interesting answer was given by interviewee G. According to him they do not spend a planned amount of time reviewing metrics and that is because they have alarms set for multiple measurements. When these alarms fall below a certain threshold then they get notified. A similar answer was also found in the secondary data where a respondent answered that they spend 0 time due to the use of triggers. Concluding, the respondents that use software quality metrics track them during code reviews using multiple tools some of which are mentioned in section 4.2. 2..

4.6 Survey results

In this subchapter we will display some of the survey's results that could help us answer the main research question. The participants of the survey were the 13 people that were also interviewed.

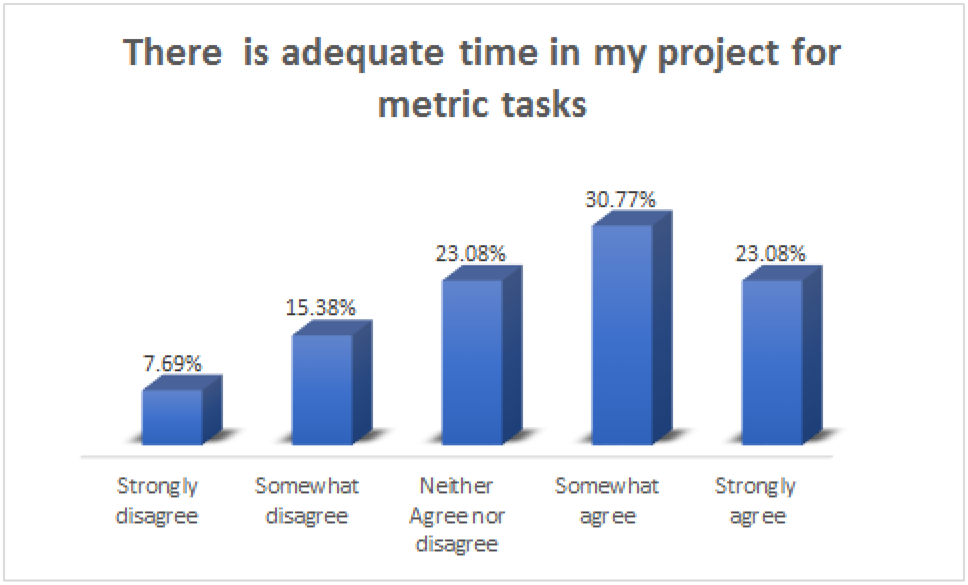


Figure 12: Time for metric tasks

The results of this figure depict that most of our interviewees believe that they have time during the implementation of their projects in order to check the metrics. This can be also combined with the answers from our interviews regarding the time spent in order to check the metrics which was 10-20% in most of the cases. However, the results of this question do not necessarily mean that the experts can make the most out of the metrics.

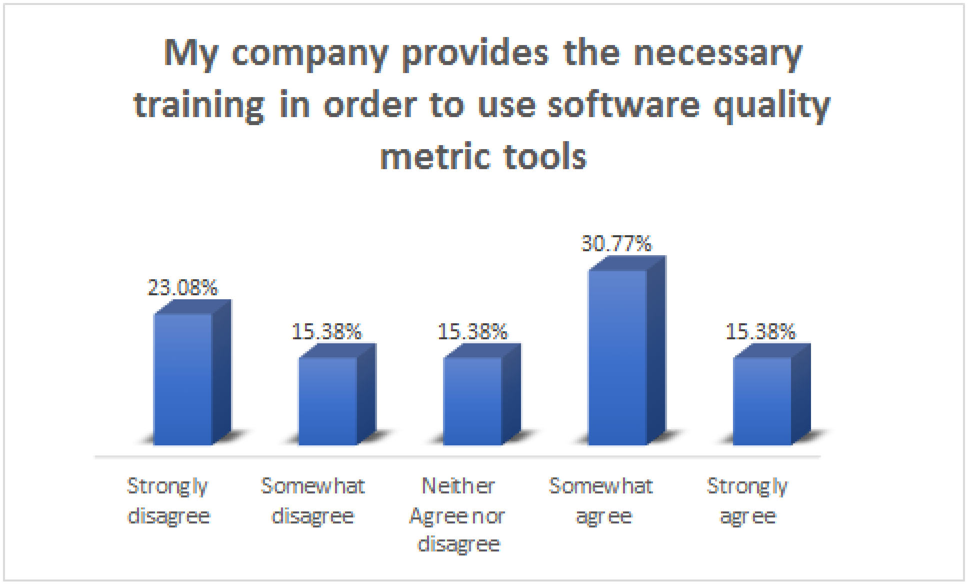


Figure 13: Training provided

As far as training is concerned, as mentioned during the interviews it is an issue of high importance regarding the use of software quality metric tools. According to the results of the survey we can say that nearly 50% of the participants' companies do not provide their employees with the appropriate training opportunities in order to get familiar with these tools and understand their relevance.

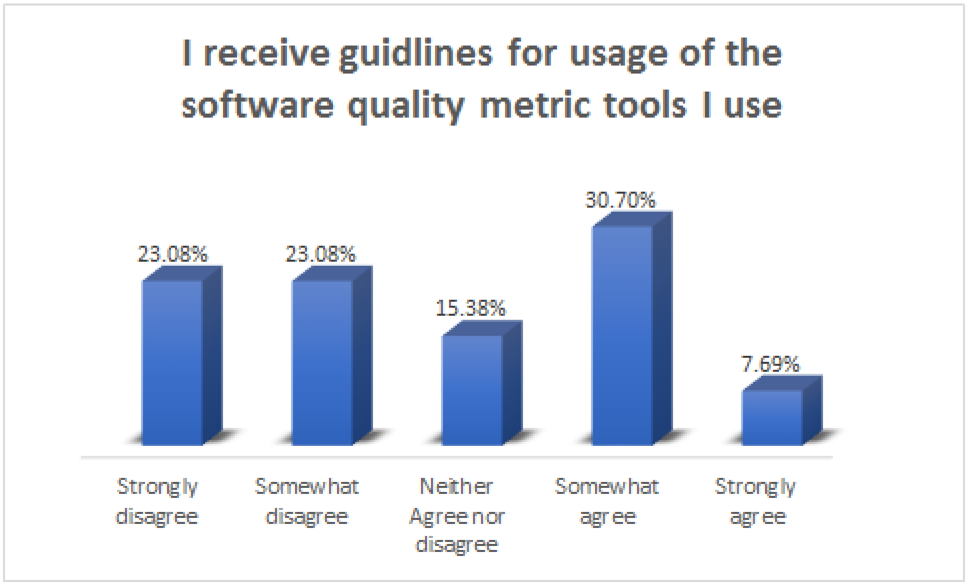


Figure 14: Guidelines for metrics usage

Similar to the training figure, this figure displays the amount of the participants that receive guidelines in order to use software quality metric tools. The results are quite similar to those from Figure 11. Consequently, almost 50% of our resource people do not receive the proper guidelines from their company in order to use software quality metrics.

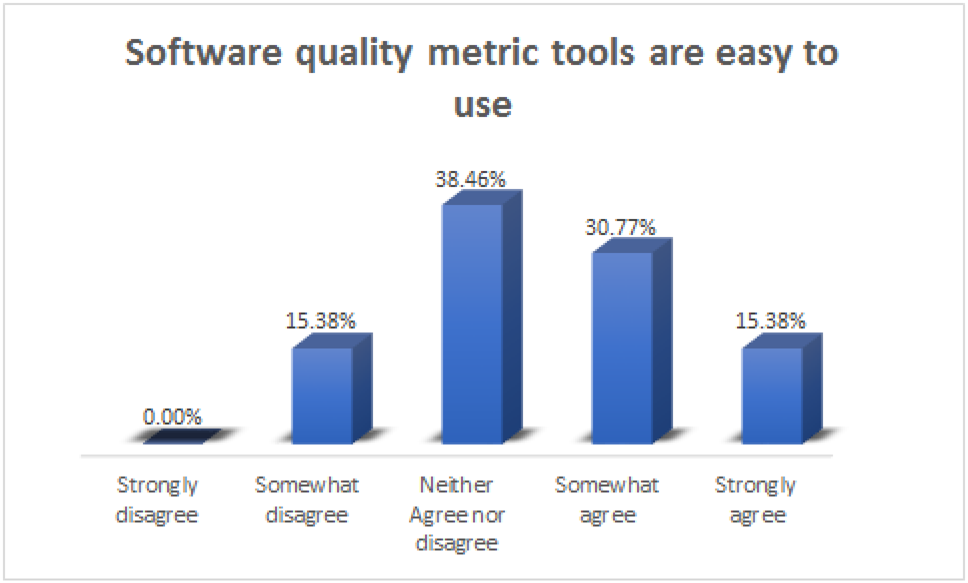


Figure 15: Ease of use for software quality metric tools

We could characterize the results of figure 13 as vague. There was a big number of respondents who described software quality metric tools as easy to use however almost 40% of the participants answered “Neither agree nor disagree. The results of this figure can be connected with the lack of training in many situations and also with the time pressure that takes place in most of the projects.

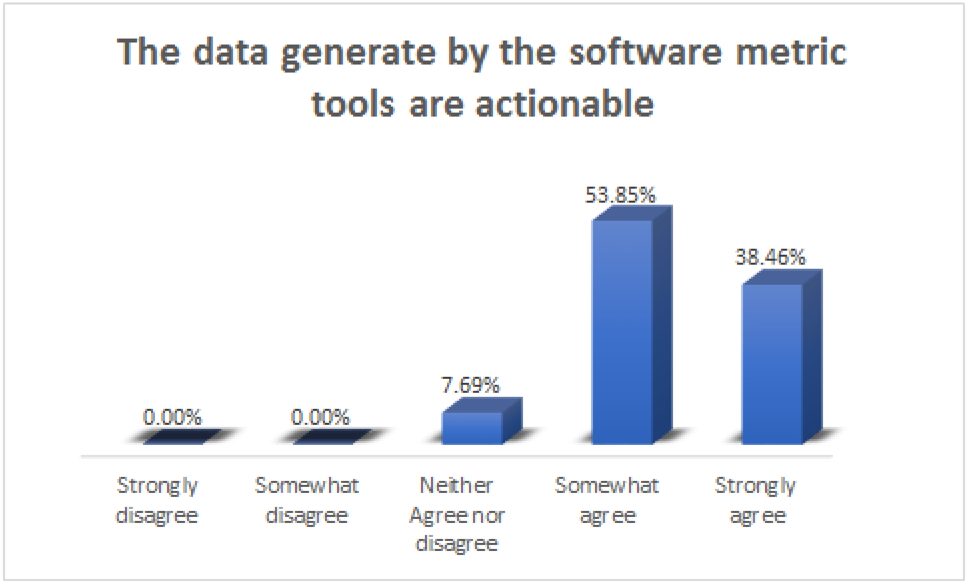


Figure 16: Actionable data from software metric tools

The results of this statement were interesting. According to the responses, most of the participants find the results derived from the metric tools actionable. This can also be combined with some answers derived from the interview from the most experienced respondents stating that software quality is changing and awareness is rising. We could say that software developers understand and value the importance of using such tools and that they believe that could positively impact the outcome of their projects.

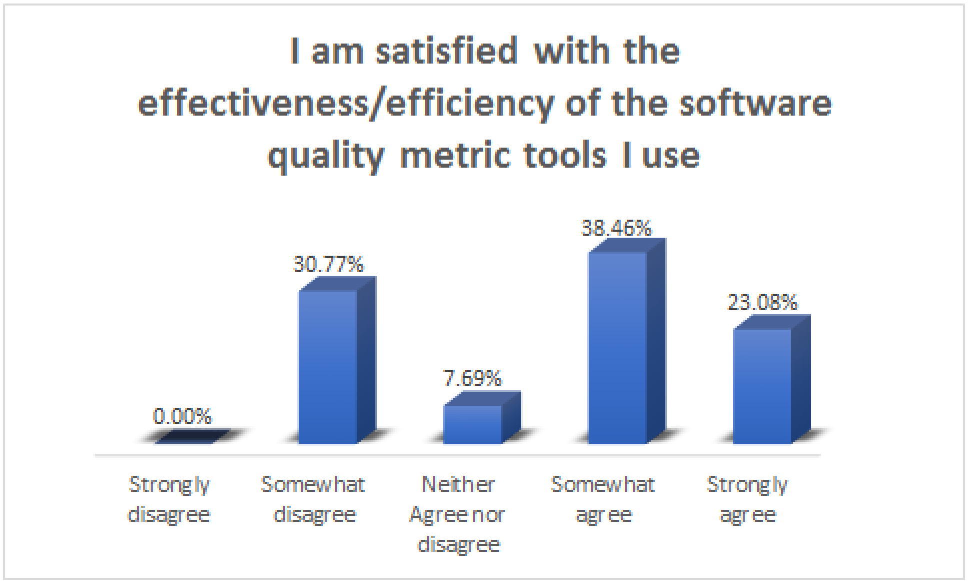


Figure 17: Satisfaction regarding software quality metric tools

Believing that the data generated by software quality metric tools are actionable (Figure 14) does not mean that those who use them are satisfied by them. In this figure there is a generous number of participants that are not satisfied by the tools used. The reasons might be plenty and could be among the obstacles and improvements, mentioned at subchapters 4.5.1 and 4.5. 2..

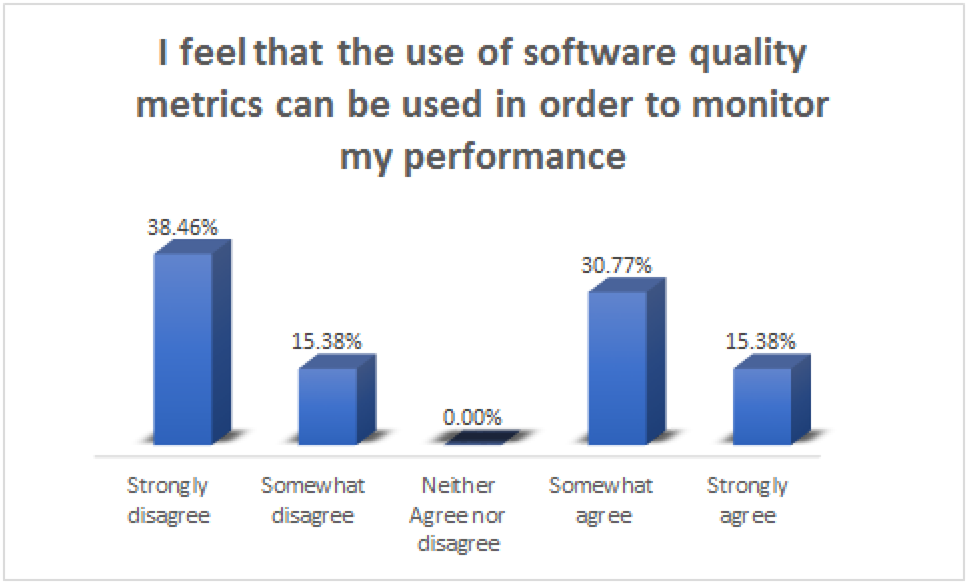


Figure 18: Performance monitoring - software metric tools

During our survey participants were asked their opinion regarding performance monitoring through software quality metrics. A percentage close to 50% believed that through metrics their performance can be monitored. This might have a negative outcome on the productivity of the software developers and it was also highlighted at the research of Fenton et al. 1998.

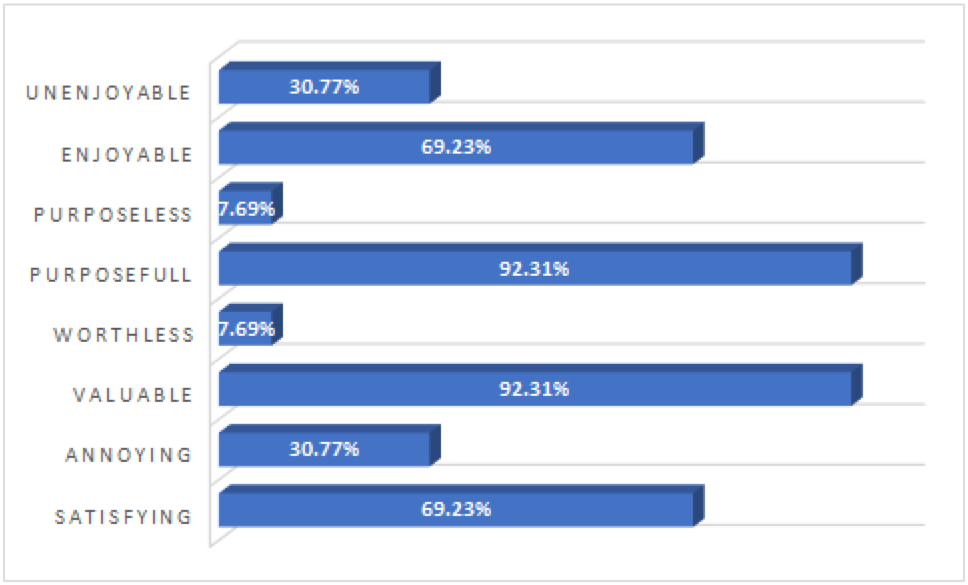


Figure 19: Perspective regarding software quality metrics

At the end of our survey we wanted to observe our participants' perspective regarding software quality metrics. It is significant to mention that 92.31% of our resource people found the metrics as purposeful and valuable. However, it is worth mentioning that 30.77% found the metrics to be unenjoyable and annoying. Hence, we can conclude that awareness regarding quality is raising, however software quality metric tools present multiple obstacles that make developers hesitant to use them.

5. Discussion

5.1 Reflection

During the previous chapter we presented the outcomes for each of our research questions. In order to achieve that, we presented the findings from our surveys/interviews and from the secondary data provided. In this section we will assess and evaluate the answers of our research questions while discussing the findings.

RSQ1: How do organizations measure their software quality?
The empirical findings for RSQ1 show that most of the organizations are becoming more aware regarding software quality. As Kitchenham (1996) argues, the increasing expectations in software market leads companies to continuously invest a significant amount of money, time and effort in order to improve their software quality. Franca and Soares (2015) highlighted the importance of software quality metrics as means in order to attain higher quality. According to our study, we have

identified that the level of awareness regarding software quality is increasing in organizations. The answers that we derived regarding how the software quality is being measured are:

- Measurements from tools
- Tests (unit, component, system)
- Customer satisfaction - ratings
- Static analysis tools
- Code checks

Consequently, the answer to RSQ1 could be the following: The level of awareness regarding software quality is increasing. A big amount of the experts highlighted the importance of high software quality and the use of multiple tools and methods in order to get insights regarding quality. Some of them were already using tools while others were in the transition of going there. On the other hand, there were also participants who stated that the fast delivery is more important from monitoring quality or that measuring quality is a slowdown in the product delivery. Hence, we can conclude that there were 3 types of categories:

1. Companies that do not measure quality at all since they are focused on releasing fast.
2. Companies that are on the transition of using tools in order to track quality.
3. Companies that have made serious steps into measuring quality through different tools.

RSQ2: What are found to be the most effective software quality metrics?
During our literature review we presented metrics found in 8 researches between 2002-2013. According to our study, we identified a set of metrics that are part of the metrics studied in the literature review. In order to answer this question, we focused on the following questions:

- What are the most important/effective software quality metrics that you use?
- When do you deem a software quality metrics as an effective one?

The metrics found are effective since they improve the outcome of a project leading to maintainable code. We gathered 18 different metrics, however, some of them had high occurrence compared to others. Consequently, the answer to the RSQ2 is that the effective metric tools should be fast and result in a maintainable product while they should be easily integrated with the development environments. The highest in occurrence software quality metrics are test coverage, lines of code, cyclomatic complexity and code duplication.

RSQ3: How does the development process impact the software quality metrics integration?
In Chapter 2 we presented research indicating the use of software quality metrics during the development processes. Kan (2002) classified the metrics in the

Traditional Development Process while Rawat (2012) further extended the findings of Kan. Oza & Korkala (2012) respectively classified the metrics of the Agile development process. Furthermore, Sfetsos & Stamelos (2010) argued that Agile development redefines quality assurance, while code reviews result in less defects and higher code quality. Hence, in order to answer this research question, we stated the next two questions.

- What type of development method do you use?
- Can the development method have an impact on software quality metrics integration?

The highest percentage of our participants used the Agile methodology and its different frameworks. Many of our interviewees had multiple years of experience in the field which means that they also experienced different methodologies in the past such as Waterfall. Consequently, the answer to the RSQ3 is that: The development process impacts everything and so does with the software quality metrics integration. Agile and its iterating processes forces teams for a continuous improvement that results in more tests and insights from software quality metrics.

Main research question: How can software quality metrics be integrated in the development process in an unobtrusive manner?

Finally, based on the on the data gathered and the 3 sub questions mentioned above, we can answer the main research question. Gopal (2012) highlighted the high complexity of the metric tools while Huisman and Livari (2006) argued that software developers do not perceive the quality of the development process as important as their managers and lacked motivation (Hall and Fenton 1997). Umarji (2009) highlighted that developers spend too much time reporting from metric tools. We can conclude that the awareness regarding software quality is increasing. Bigger companies are already using metrics while even more small or medium ones are using tools in order to get valuable insights regarding their software. The majority of our interviewees found the use of metrics significant in order to produce a high maintainable product. Consequently, in order that the software quality metrics can be integrated in the development process companies should raise the awareness regarding software quality and transfer the value of quality into their development teams. Trainings regarding with the right use of metric tools should be arranged. By offering extensive trainings, software developers might make the most out of the insights provided leading to a high maintainable code. Moreover, by gaining extensive experience regarding these tools, developers will spend less time in order to translate the insights of the metrics. Furthermore, it would be meaningful if the metric tools could be better integrated with the development environments used by the developers so that the teams can have fast and accurate results. Last but not least, the community behind each tool could be bigger and better supported in order to solve the issues addressed by its users.

5.2 Software quality measurement

During our data gathering we tried to identify how organizations measure software quality. The question asked was: “How would you describe the quality of your software? How do you measure it? The first part of the question was barely answered by our interviewees indicating that the evaluation of software quality is an ambiguous process. This was also stated by Hendriks et al. (2000). People seemed to be unaware regarding the strategy their organization was following for software quality. However, most of the participants mentioned that they make use of multiple tools, methods and metrics in order to measure software quality.

5.3 Software quality metrics in practice

In section 4.2.2 and 4.3.1. We presented the methods and tools collected. According to the data gathered, this study collected 19 metrics most of which were reviewed in the literature at chapter 2.9. Moreover, except from the mentioned metrics, organizations are using specific tools in order to manage quality. These tools provide users with insights regarding quality and metrics. There is a big variety of tools used such as SonarQube and Jira (chapter 4.2.2). Based on 13 interviews/surveys and the secondary data collected and analyzed, the general impression is that awareness is increasing regarding software quality metrics and their collection.

According to figure 15, almost 92% of our interviewees agreed that the data generated by the metric tools are actionable. However, in the next figure, figure 16, almost 60% of them is seem not to be satisfied by the effectiveness/efficiency of the tools that they are using. Similarly, at figure 16, 30% of the participants found the software quality metrics unenjoyable and annoying during their work. There might be various reasons behind those answers, some of which are stated at the obstacles part in section 4.5.1.. Consequently, time pressure and lack of training might influence the perspective of the developers regarding metrics. The functionality and the integration issues of these tools might also play a significant part behind those percentages.

As far as the performance monitoring is concerned, nearly 45% of our participants agreed that through the use of metric tools their performance might be used in order to monitor their performance. As stated in the literature review, Fenton et al. (1998) were the first that highlighted that issue during their research.

However, according to our respondents most of the organizations gather and rely on metrics that are crucial for them. We gathered and analyzed the most significant metrics according to our respondents. The results demonstrate that Test Coverage (26), Lines of code (14), Cyclomatic Complexity (12) and Code Duplication (6) are the metrics with the highest occurrence among our participants and their organizations. Last but not least, we observed that besides the interrelation between traditional and agile metrics, the traditional are the ones used the most.

5.4 Software quality metrics integration

From the data we gathered both from the interviews//surveys and from the secondary data we managed to get familiar with some of the obstacles and the improvements our participants would expect from the tools they use in order to track metrics. The time spent reviewing metrics fluctuates from 0%-30% for our resource people. There was a variety of people from whom some used metrics constantly, some did not review them due to the fact they had alarms set for measurements while others did not review them at all. From the participants that use metrics the most frequent obstacle mentioned was the tools were poorly integrated with their development environments resulting in time issues while reviewing the metrics. Moreover, lack of guidance was also mentioned as an obstacle. More specifically the community behind the tools used, seemed to be weak according to 3 interviewees making it hard to understand and interpret the numbers derived from the metric tools. These reasons in combination with the lack of training often makes the use of metrics a time-consuming process for the development teams.

5.4 Problem definition

After analyzing the data from 13 interviews/surveys and the secondary data provided we can say that software quality metrics integration in the development process is a complex task that can be affected by various variables. We can conclude that the basic pillars of the problematic integration can be further categorized into two classes, which are the organizations and the tools.

Organizations: It is evident from the results, that many organizations do not have a clear plan regarding how to measure software quality. They seemed aware of the potential benefits of implementing metrics, however, there was lack of a clear plan on how to implement them in the development process and get the most out of them. Our interviewees stated that lack of training was the most important issue in combination with the restricted time in order to review the metrics. Furthermore, none of them stated a clear plan nor a defined strategy regarding metrics.

Tools: Apart from the problems that exist in the organizations regarding software quality metrics through our study we identified that people involved with software delivery do not find the tools used in order to attain higher software quality fully functional. Software practitioners that are using these tools should be able to rely on them in order to get the appropriate measurements that would lead to higher software quality.

6. Conclusion

During this research, the author took a deep look into the topic of software quality. The focus was on the software quality metrics and more specifically on their integration in the development process. For the purpose of our research 13 people from different organizations were interviewed while the researcher also used secondary data regarding the subject. All of our resource persons were involved with software delivery while their experience varied between 1-30 years.

Further in this section we will draw a conclusion including the main findings regarding software quality metrics. Furthermore, the researcher will discuss the limitations and the proposals for further research.

6.1 Recommendations

Software quality metrics are considered to be a vital part of software engineering as the software industry grows. It is believed that through them software engineering and management practices will be improved. Software engineers and managers are in need of better understanding of their software development process so that to make the appropriate changes in order to improve productivity and quality. Through metrics, progress can be measured while at the same time any possible risks can be mitigated, lowering costs and improving quality. Moreover, as pointed out in Chapter 2, the use of software metrics has a high potential payoff but it is a fraught process. Besides, multiple organizations are moving toward metric programs. However, functionality is the top priority and quality comes next. Even though our interviews were anonymous, we can state that resource persons from well-known multinational companies were implementing metric programs and were very knowledgeable about them. This might start a bandwagon effect among small and medium size companies and raise the general awareness of metrics.

Through our paper we tried to identify the metrics experience of multiple people working within software delivery. We introduced empirical validation of some measures and tools that companies use in order to measure software quality and we tried to identify the possible bottlenecks of not making the use of these tools an enjoyable process for our interviewees. We believe that the results of our study can be used in order to provide with valuable insights both practitioners and researchers of the field.

Taking into consideration the results part of Chapter 4, we would like to state some recommendations that were resulted by the literature review and our data analysis.

Organizations: The importance of the metrics is undeniable. Hence, any organization that is aware of software quality should assess and understand its measurement capabilities and design a process upon these. Consequently, a software quality strategy should be defined and communicated accordingly to the employees. That is because everyone that is part of a development team is accountable for the quality of the end product. As a result, the software quality strategy should always be the core of the development process and a top priority for the Business & IT teams before initializing any project. The Project Manager, the Architects, the Testers and all the key team members should be aware and define this strategy. The software quality strategy should cover all the crucial quality objectives that take place during the construction phase of a project such as: test cases and planning, code reviews and metrics identification. There should be a hierarchy of tests during the construction phase with the use of metrics in every level. Furthermore, in order to further raise the awareness of the software quality metrics benefits, it is important to potentially have an improved communication between the people involved with software delivery. As addressed in our results, it is meaningful to implement educational programs in order to potentially increase the right use of metrics. The education and training programs should provide a clear overview of the advantages of the metrics in the development process. These programs should be continuously arranged in order to get the teams familiar with the recent trends and tools that exist. By highlighting the usefulness of the metric tools and their insights, developers' enthusiasm would possibly be raised while collecting the metrics. Then, it is crucial that people who collect the metrics should understand the data collected, why they are collected and how it can be used in order to add value to the end product. So, the metric programs -if applied- should be transparent and obvious to all the persons involved. Moreover, the design of the metric programs should involve developers whose viewpoint should be taken into serious consideration at the design stage. Furthermore, by providing feedback to the data collected, developers will get a clear indication that the data they collect is being used. Another meaningful action would be to create metric teams that would be responsible for the metric programs and to assign tasks to specific individuals. In that way metric programs are assigned to dedicated teams who are responsible for the implementation. These teams might act as the liaison between the managers, developers and the users in order to provide them with a better understanding regarding software metrics and propose the most suitable for each occasion. Last but not least it is really crucial that companies collect specific data for specific purposes. Gathering metrics without using them is a time-consuming process that puts extra work pressure to the teams. Related to that, Caldiera et al. (1994) implemented the goal-question-metric approach. According to this model a goal has to be defined which will lead to a set of questions in order to characterize the way to achieve the goal, leading to a set of metrics associated with every question in order to answer it in a quantitative way.

Tools: According to our responses, we can say that a large amount of the developers using metric tools are not satisfied by its functionalities. As a result, there is need for more intelligent tools. By utilizing more intelligent tools, extra work will be minimized for the developers and possibly their resistance to the use of metrics. It is evident that the technological developments will also affect the metric tools and possibly in the near future we might see multiple well-designed tools. It would be meaningful if these tools might be able to provide a smooth integration with a variety of coding environments with a large community and guidance in order to solve the problems that the users might face.

Hence, we can say that the success of metric programs depends on a well-planned strategy and the use of the right tools. In such a strategy, functionality and quality should be implemented together. Moreover, we find it meaningful to provide some steps according to the experience gained after the readings and from the data collected.

1. **Identify the persons** who will make decisions/actions based on the metrics. (managers, testers, software engineers etc.). There are multiple metrics so the organizations should collect the ones that are needed by specific people and not to collect, report and analyze metrics if no one is using them.
2. **Set goals** while listening to the customers' requirements that could be useful in order to select the appropriate metrics. Software quality metrics should provide with information in order to manage and improve the software development processes. More specifically the metrics can be the way in order to reach the goals more effectively.
3. **Select the appropriate metrics** that are the ones that can satisfy the customers' requirements. Software quality metrics will indicate and provide information so that experts provide more informed decisions in order to produce a maintainable product.
4. **Operationalize software definitions**. It is important that everyone within the team has the same belief of a term like software quality or software maintainability. Different interpretations by people regarding these entities may lead to difficulties in understanding the significance of software quality metrics usage.
5. **Provide comprehensive education and training programs** within organizations so that the awareness of the benefits of both software quality and the metric tools is increased.

6.2 Validity

Due to the nature of this research, there is space for validity concerns. Firstly, there were time and resources limitations. More specifically, the time and the resources of the research was limited into 13 interviews while the research lasted 6 months. Moreover, there was use of secondary data, including interviews that were conducted by multiple people and not from the researcher. Hence, in order to get a better understanding regarding software quality and the metrics, there is a need to interview more people from each company including companies from different sectors. As far as the coding is concerned, it was performed only by the author and as a result its subjectivity can be questioned

6.3 Recommendations for future research

The main question explored during this research was “How can software quality metrics be integrated in the development process in an unobtrusive manner?”. For this purpose, we interviewed 13 experts working in the software development field while we also analyzed secondary data relevant to our topic. Thus, there is need for further research with more organizations of multiple industries including more than 1 person per case. This will provide a clearer view of how people and organizations value software quality and metrics. Consequently, any future research should extend to a larger scale, meaning the number of participants for the interviews and the survey should be greatly increased.

Finally, according to our study, software quality and software quality measurement seems to be vague in organizations. Consequently, based on our research results we concluded that apart from a larger scale research on a similar topic, it would be interesting to conduct further research in the field of the strategy within the companies regarding software quality and a comparative research regarding the tools used nowadays.

7. References

Baker, B. S. (1995, July). On finding duplication and near-duplication in large software systems. In Proceedings of 2nd Working Conference on Reverse Engineering (pp. 86-95). IEEE.

Beck, K., & Gamma, E. (2000). Extreme programming explained: embrace change. addison-wesley professional.

Bouwers, E., van Deursen, A., & Visser, J. (2013, May). Evaluating usefulness of software metrics: an industrial experience report. In Software Engineering (ICSE), 2013 35th International Conference on (pp. 921-930). IEEE

Bijlsma, D., Ferreira, M. A., Luijten, B., & Visser, J. (2012). Faster issue resolution with higher technical quality of software. Software quality journal, 20(2), 265-285.

Briand, L. C., Morasca, S., & Basili, V. R. (1999). Defining and validating measures for object-based high-level design. IEEE transactions on software engineering, 25(5), 722-743.

Boehm, B. W., Brown, J. R., & Lipow, M. (1976, October). Quantitative evaluation of software quality. In Proceedings of the 2nd international conference on Software engineering (pp. 592-605). IEEE Computer Society Press.

Caldiera, V. R. B. G., & Rombach, H. D. (1994). The goal question metric approach. Encyclopedia of software engineering, 528-532.

Chawla, S., & Kaur, G. (2013). Comparative Study of the Software Metrics for the complexity and Maintainability of Software Development. International Journal of Advanced Computer Science & Applications, 4.

Chang, S. K. (2001). Handbook of software engineering and knowledge engineering (Vol. 1). World Scientific.

Corbin, J. M., & Strauss, A. (1990). Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative sociology, 13(1), 3-21.

Czerwonka, J., Greiler, M., & Tilford, J. (2015, May). Code reviews do not find bugs: how the current code review best practice slows us down. In Proceedings of the 37th International Conference on Software Engineering-Volume 2(pp. 27-28). IEEE Press.

Dingsøyr, T., Dybå, T., & Abrahamsson, P. (2008). A preliminary roadmap for research on agile software development research. In Proc. Agile Conference (pp. 83-96).

Downey, S., & Sutherland, J. (2013, January). Scrum metrics for hyperproductive teams: how they fly like fighter aircraft. In 2013 46th Hawaii International Conference on System Sciences (pp. 4870-4878). IEEE.

Dutta, B. (2009). Enterprise Software Metrics: How To Add Business Value (Doctoral dissertation, Kent State University).

Elgebeely, A. R. (2014). Software quality challenges and practice recommendations. IBM. http://www. ibm. com/developerworks/rational/library/software-quality-challenges-practice-recommendations/. Accessed, 15.

Ewusi-Mensah, K. (1997). Critical issues in abandoned information systems development projects. Communications of the ACM, 40(9), 74-80.

Fenton, N. E., & Pfleeger, S. L. (1998). Software Metrics: A Rigorous and Practical Approach: Brooks.

Fenton, N. E., & Neil, M. (1999). Software metrics: successes, failures and new directions. Journal of Systems and Software, 47(2-3), 149-157.

Finkelstein, L., & Leaning, M. S. (1984). A review of the fundamental concepts of measurement. Measurement, 2(1), 25-34.

França, J. M., & Soares, M. S. (2015). SOAQM: Quality Model for SOA Applications based on ISO 25010. In ICEIS (2) (pp. 60-70).

Garvin, D. A. (1984). What Does “hltoduct Quality” Really Mean. Sloan management review, 25.

Glass, R. L. (2002). Facts and Fallacies of Software Engineering: FREQ FORGOT FUND FACTS _p1. Addison-Wesley Professional.

Glazer, H., Dalton, J., Anderson, D., Konrad, M., & Shrum, S. C. (2008). or Agile: Why Not Embrace Both?’, Software Engineering Institute.

Ghanam, Y., & Carpendale, S. (2008). A survey paper on software architecture visualization. University of Calgary, Tech. Rep.

Gopal, A., Krishnan, M. S., Mukhopadhyay, T., & Goldenson, D. R. (2002). Measurement programs in software development: determinants of success. IEEE Transactions on software engineering, 28(9), 863-875.

Hall, T., & Fenton, N. (1997). Implementing effective software metrics programs. IEEE software, 14(2), 55-65.

Heitlager, I., Kuipers, T., & Visser, J. (2007, September). A practical model for measuring maintainability. In 6th international conference on the quality of information and communications technology (QUATIC 2007) (pp. 30-39). IEEE.

Huisman, M., & Iivari, J. (2006). Deployment of systems development methodologies: Perceptual congruence between IS managers and systems developers. Information & Management, 43(1), 29-49.

Imreh, R., & Raisinghani, M. (2011). Impact of agile software development on quality within information technology organizations. Journal of Emerging Trends in Computing and Information Sciences, 2(10), 460-475.

IEEE Std. 610.12-1990. Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA, 1993.

IEEE. (1998). IEEE Std 1074 -1997 - Standard for Software Life Cycle Processes

IEEE Standards Association et al. Ieee standard for a software quality metrics methodology. IEEE Std, pages 1061–1998, 1998.

International Organization for Standardization, & International Electrotechnical Commission. (2001). Software Engineering--Product Quality: Quality model (Vol. 1). ISO/IEC.

Kan, S. H. (2002). Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc..

Kasurinen, J., Taipale, O., Vanhanen, J., & Smolander, K. (2012). Exploring the perceived end-product quality in software-developing organizations. International Journal of Information System Modeling and Design (IJISMD), 3(2), 1-32

Kassim, N. M., & Zain, M. (2004). Assessing the measurement of organizational agility. Journal of American Academy of Business, Cambridge, 4(1/2), 174-177.

Kitchenham, B., & Pfleeger, S. L. (1996). Software quality: the elusive target [special issues section]. IEEE software, 13(1), 12-21.

Kunz, M., Dumke, R. R., & Zenker, N. (2008, March). Software metrics for agile software development. In 19th Australian Conference on Software Engineering (aswec 2008) (pp. 673-678). IEEE.

Luijten, B., Visser, J., & Zaidman, A. (2010, March). Faster defect resolution with higher technical quality of software. In 4th international workshop on software quality and maintainability (SQM 2010).

Lundberg, L., Mattsson, M., & Wohlin, C. (2005). Software quality attributes and trade-offs. Blekinge Institute of Technology, Karlskrona.

Leffingwell, D. (2016). SAFe® 4.0 Reference Guide: Scaled Agile Framework® for Lean Software and Systems Engineering. Addison-Wesley Professional.

McDermid, J. A. (Ed.). (2013). Software engineer's reference book. Elsevier.

Miguel, J. P., Mauricio, D., & Rodríguez, G. (2014). A review of software quality models for the evaluation of software products. arXiv preprint arXiv:1412.2977.

Mordal, K., Anquetil, N., Laval, J., Serebrenik, A., Vasilescu, B., & Ducasse, S. (2013). Software quality metrics aggregation in industry. Journal of Software: Evolution and Process, 25(10), 1117-1135.

Oza, N., & Korkala, M. (2012, March). Lessons Learned In Implementing Agile Software Development Metrics. In UKAIS (p. 38).

Olatunji, S. O., Rasheed, Z., Sattar, K. A., Al-Mana, A. M., Alshayeb, M., & El-Sebakhy, E. A. (2010). Extreme learning machine as maintainability prediction model for object-oriented software systems. Journal of Computing, 2(8), 49-56.

Oman, P., & Hagemeister, J. (1994). Construction and testing of polynomials predicting software maintainability. Journal of Systems and Software, 24(3), 251-266.

Orlikowski, W. J., & Gash, D. C. (1994). Technological frames: making sense of information technology in organizations. ACM Transactions on Information Systems (TOIS), 12(2), 174-207.

Pfleeger, S. L. (2008). Software metrics: Progress after 25 years?. IEEE Software, 25(6), 32-34.

Paramshetti, P., & Phalke, D. A. (2014). Survey on software defect prediction using machine learning techniques. International Journal of Science and Research (IJSR), 3(12), 1394-1397.

Penny, G. (2003). Software maintenance: concepts and practice. World Scientific.

Punia, S. K., Kumar, P., & Gupta, A. (2016, August 8). A Review of Software Quality Metrics for Object-Oriented …

Purcell, J. E. (2007, February). Comparison of software development lifecycle methodologies. In Web Application Security Workshop, White Paper (February 2007).

Pulford, K., Kuntzmann-Combelles, A., & Shirlaw, S. (1995). A quantitative approach to software management: the AMI handbook. Addison-Wesley Longman Publishing Co., Inc..

Rawat, M. S., Mittal, A., & Dubey, S. K. (2012). Survey on impact of software metrics on software quality. IJACSA) International Journal of Advanced Computer Science and Applications, 3(1).

Rentrop, J. (2006). Software metrics as benchmarks for source code quality of software systems. Vrije Universiteit, Amsterdam.

Saraiva, J., Soares, S., & Castor, F. (2013, May). Towards a catalog of object-oriented software maintainability metrics. In 2013 4th International Workshop on Emerging Trends in Software Metrics (WETSoM) (pp. 84-87). IEEE.f

Sato, D., Goldman, A., & Kon, F. (2007, June). Tracking the evolution of object-oriented quality metrics on agile projects. In International Conference on Extreme Programming and Agile Processes in Software Engineering (pp. 84-92). Springer, Berlin, Heidelberg.

Strauss, A., & Corbin, J. (1994). Grounded theory methodology. Handbook of qualitative research, 17, 273-85.

Scotto, M., Sillitti, A., Succi, G., & Vernazza, T. (2006). A non-invasive approach to product metrics collection. Journal of Systems Architecture, 52(11), 668-675.

Sheetz, S. D., Henderson, D., & Wallace, L. (2009). Understanding developer and manager perceptions of function points and source lines of code. Journal of Systems and Software, 82(9), 1540-1549.

Sfetsos, P., & Stamelos, I. (2010, September). Empirical studies on quality in agile practices: A systematic literature review. In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the (pp. 44-53). IEEE.

Umarji, M., & Seaman, C. (2009, October). Gauging acceptance of software metrics: Comparing perspectives of managers and developers. In 2009 3rd International Symposium on Empirical Software Engineering and Measurement (pp. 236-247). IEEE.

Van Veenendaal, E., Hendriks, R., & Van Vonderen, R. (2002). Measuring software product quality. Software quality professional, 5(1), 6.

Visser, J. (2015). SIG/TÜViT evaluation criteria trusted product maintainability: Guidance for producers. Software Improvement Group, Tech. Rep., 7.

Visser, I. J. (2018). SIG/TÜViT Evaluation Criteria Trusted Product Maintainability: Guidance for producers Version 10.0. Software Improvement Group.

8. Appendix

8.1 Software quality metrics – literature

Metric Names	Rawat et al. (2012)	Chawla & Kaur (2013)	Kan (2012)	Saraiva et al. (2013)	Sfetsos & Stamelos (2010)	Kunz et al. (2008)
Source lines of code (SLOC)	✓	✓				
Function Point (FP)	✓					
Object-Oriented (OO)	✓					
Comment Percentage (CP)		✓				
Halstead for complexity measurement (HM)		✓			✓	
Cyclomatic Complexity (CC)		✓		✓	✓	
Weighted Method per Class (WMC)		✓		✓	✓	
Depth of Inheritance Tree (DIT)		✓		✓		✓
Number of Children (NOC)		✓		✓	✓	✓
Coupling Between Objects (CBO)		✓		✓		✓
Response of a Class (RFC)		✓		✓		
Lack of Cohesion (LCOM)		✓		✓	✓	
Lines of Code (LOC)				✓		✓

Object Oriented Design (MOOD)		✓				
Defect Density (DD)			✓			
Mean Time to Failure (MTTF)			✓			
Customer Problems			✓			
Customer Satisfaction			✓			
Defect Arrival			✓			
Phase-Based Defect Removal Pattern			✓			
Fix Backlog			✓			
Backlog Management Index			✓			
Fix Response Time			✓			
Percent Delinquent Fixes			✓			
Fix Quality			✓			
Number of Associations (NASSOC)				✓		
Number of Aggregations (NAGGH)				✓		
Number of generalization (NGENH)				✓		
Number of Classes (NC)				✓	✓	✓
Number of Methods (NMC)				✓		
Message Passing Coupling (MPC)				✓		
Weighted Attributes per Class (WAC)				✓		
Degree of Cohesion (DC)				✓		

Coupling Through Abstract Data (CTA)				✓		
Comment Ration (CR)				✓		
Number of Name-Parts (NNP)						✓
Number of Comment-Lines (CL)						✓
Number of Local Variables (NLV)						✓
Number of Created Objects (NCO)						✓
Number of Referring Objects (NRO)						✓
Number of Parameters (NP)						✓
Cycle Count of Dependency Graph (CCDG)						✓

8.2 Survey sample

1. There is adequate time in my project for metrics tasks.

   ○ Strongly disagree

   ○ Somewhat disagree

   ○ Neither agree nor disagree

   ○ Somewhat agree

   ○ Strongly agree

2. My company provides the necessary training in order to use software quality metric tools.

   ○ Strongly disagree

   ○ Somewhat disagree

   ○ Neither agree nor disagree

   ○ Somewhat agree

   ○ Strongly agree

3. People knowledgeable about metrics, are easy to approach within my company.

   ○ Strongly disagree

   ○ Somewhat disagree

   ○ Neither agree nor disagree

   ○ Somewhat agree

   ○ Strongly agree

4. I receive guidelines for usage of the software quality metric tools I use.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

5. Using and reporting data from software quality metrics tools takes too much time.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

6. Learning how to use software quality metrics tools was a tedious task.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

7. Software quality metrics tools are easy to use.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

8. Too much mental effort is needed in order to report data derived from the software quality metric tools.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

9. The data generated by the metrics are actionable.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

10. What metrics do you currently collect from the ones listed below? Please mention any software quality metrics that you use that are not in the list.

- ☐ Lines of Code
- ☐ Cyclomatic Complexity
- ☐ Lack of Cohesion
- ☐ Weighted Method per Class
- ☐ Number of Children
- ☐ Defect Density
- ☐ Other: ____

11. My team members are enthusiastic about software quality metrics.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

12. I am satisfied with the effectiveness/efficiency of the software quality metric tools I use.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

13. The software quality metric tools I use are well-integrated with the other tools used (in terms of workflow, data format etc.).

○ Strongly disagree

○ Somewhat disagree

○ Neither agree nor disagree

○ Somewhat agree

○ Strongly agree

14. Using software quality metrics during my projects is:

○ Satisfying

○ Annoying

○ Valuable

○ Worthless

○ Purposeful

○ Purposeless

○ Enjoyable

○ Unenjoyable

15. I feel that the use of software quality metrics can be used in order to monitor my performance.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

16. I agree with employers using software quality metrics in order to evaluate employee performance.

- ○ Strongly disagree
- ○ Somewhat disagree
- ○ Neither agree nor disagree
- ○ Somewhat agree
- ○ Strongly agree

8.3 Interview questions

1.What is your role and responsibility within the company?

2. How many years of software development experience do you have?

3. What is your current assignment and from how many people does your team consists of?

4. How would you describe the quality of your software? How do you measure it?

5. What type of development method do you use?

6. Can the development method have an impact on software quality metrics integration? Can you elaborate?

7. What methods or tools does your team use in order to improve software quality and why?

8. How much has the use of metrics improved your projects' performance?

9. How much time do you spend on reviewing software quality metrics (percentage)?
Can you elaborate?

10. Is there time available to fix complexity or duplication issues in your code? What is its percentage over your total working time?

11. What are the obstacles in your opinion for the quality metrics to be easily integrated?

12. What kind of improvements would you expect in these tools?

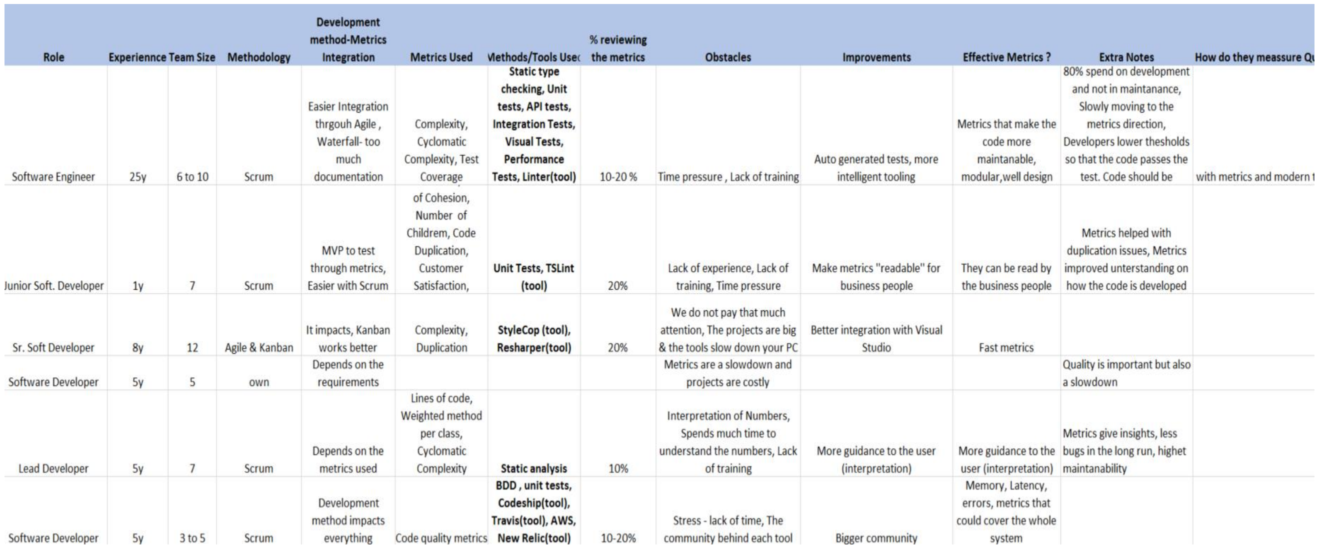
13. What are the most effective software quality metrics that you use? Why these?

14. When do you deem a software metric as an effective one?

15. Is there any other information that you would like to share with me?

16. Are you available for follow- questions

8.4 Interview transcript sample



Role	Experiennce	Team Size	Methodology	Development method-Metrics Integration	Metrics Used	Methods/Tools Used	% reviewing the metrics	Obstacles	Improvements	Effective Metrics ?	Extra Notes	How do they meassure Q
Software Engineer	25y	6 to 10	Scrum	Easier Integration thrgouh Agile , Waterfall- too much documentation	Complexity, Cyclomatic Complexity, Test Coverage	**Static type checking, Unit tests, API tests, Integration Tests, Visual Tests, Performance Tests, Linter(tool)**	10-20 %	Time pressure , Lack of training	Auto generated tests, more intelligent tooling	Metrics that make the code more maintanable, modular,well design	80% spend on development and not in maintanance, Slowly moving to the metrics direction, Developers lower thesholds so that the code passes the test. Code should be	with metrics and modern t
Junior Soft. Developer	1y	7	Scrum	MVP to test through metrics, Easier with Scrum	of Cohesion, Number of Childrem, Code Duplication, Customer Satisfaction,	**Unit Tests, TSLint (tool)**	20%	Lack of experience, Lack of training, Time pressure	Make metrics "readable" for business people	They can be read by the business people	Metrics helped with duplication issues, Metrics improved understanding on how the code is developed	
Sr. Soft Developer	8y	12	Agile & Kanban	It impacts, Kanban works better	Complexity, Duplication	**StyleCop (tool), Resharper(tool)**	20%	We do not pay that much attention, The projects are big & the tools slow down your PC	Better integration with Visual Studio	Fast metrics		
Software Developer	5y	5	own	Depends on the requirements				Metrics are a slowdown and projects are costly			Quality is important but also a slowdown	
Lead Developer	5y	7	Scrum	Depends on the metrics used	Lines of code, Weighted method per class, Cyclomatic Complexity	**Static analysis**	10%	Interpretation of Numbers, Spends much time to understand the numbers, Lack of training	More guidance to the user (interpretation)	More guidance to the user (interpretation)	Metrics give insights, less bugs in the long run, highet maintanability	
Software Developer	5y	3 to 5	Scrum	Development method impacts everything	Code quality metrics	**BDD , unit tests, Codeship(tool), Travis(tool), AWS, New Relic(tool)**	10-20%	Stress - lack of time, The community behind each tool	Bigger community	Memory, Latency, errors, metrics that could cover the whole system		