Software Quality Assessment in an Agile Environment

Master's Thesis

Radboud Universiteit Nijmegen

Falk Martin Oberscheven

Software Quality Assessment in an Agile Environment

Master's thesis

submitted to the Faculty of Science
of Radboud University in Nijmegen
for the degree of

Master of science

in

computing science

by

Falk Martin Oberscheven
born in Düsseldorf, Germany

Software Quality Assessment in an Agile Environment

Author: Falk Martin Oberscheven
Student Number: s0721956
E-mail: M.Oberschev@arcor.de

Abstract

Continuous improvement is a key factor for survival in today's turbulent business environment. This counts for civil engineering and even more so in the fast-paced world of information technology and software engineering. The agile methodologies, like scrum, have a dedicated step in the process which targets the improvement of the development process and software products. Crucial for process improvement is to gain information which enables you to assess the state of the process and its products. From the status information you can plan actions for improvement and also assess the success of those actions. This study constructs a model which measure the software quality of the development process. The software quality is dependent on the functional and structural quality of the software products, as well as the quality of the development process itself. Functional quality covers the adherence to user requirements, whereas the structural quality addresses the structure of the software product's source code concerning its maintainability. The process quality is related to the reliability and predictability of the development process. The software quality model is applied in a business context by collecting the data for the software metrics in the model. To evaluate the software quality model we analyze the data and present it to the people involved in the agile software development process. The results from the application and the user feedback suggest that the model enables a fair assessment of the software quality and that it can be used to support the continuous improvement of the development process and software products.

Keywords Software metrics, Software Quality, Agile software development.

University Supervisors: Dr. Arjan van Rooij
Dr. Joost Visser
Company Supervisors: André Grothues
Kai Müller

Contents

Contents **II**

1 Introduction **1**
1.1 Problem Statement 1
1.2 Research Questions 2
1.3 Research Context 3
1.4 Research Method 4
1.5 Thesis Outline 5

2 Background **6**
2.1 Agile Methodologies 6
2.2 Scrum 8
2.3 Issue Tracking Systems 9

3 Constructing a Software Measurement Model **11**
3.1 Prerequisites for Software Metrics 11
3.2 Approach to Define a Software Measurement Model 13
3.3 Linking Business Goals To Measurement Goals 14
3.4 Method to Calibrate Metric Thresholds 20

4 Software Quality Model for Agile Environments **24**
4.1 Overview 24
4.2 Goals 27
4.3 Questions 28
4.4 Metrics 34

5 Evaluation of the Software Quality Model **41**
5.1 Model Calibration 41
5.2 Model Application 45
5.2.1 Evaluation of Sprint Performance 45
5.2.2 Tracking of Team Performance 48

5.2.3 Team Comparison 49
5.3 Model Evaluation 52
5.3.1 User Feedback Questionnaire 52
5.3.2 Results 54

6 Conclusion and Future Work 57
6.1 Conclusions 57
6.2 Contributions 60
6.3 Threats to Validity 61
6.3.1 Construct Validity 61
6.3.2 Internal Validity 62
6.3.3 External Validity 63
6.4 Future Work 64

Bibliography 67

Glossary 71

A Software Quality Model (GQM) 73

B Feedback Questionnaire 75

Chapter 1

Introduction

1.1 Problem Statement

Software engineering is defined as "...the discipline of developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them" [1]. Today, software engineering is faced with rapid change in user requirements, technology. To cope with the rapid change in the business environment, the software development process must be assessed and adapted frequently. According to Fowler, the traditional methodologies or engineering methodologies, like the waterfall methodology, are not suitable for environments with rapid change, "Engineering methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change" [20].

Agile methodologies evolved to promote continuous improvement. Scrum belongs to the agile methodologies and defines frequent feedback loops for improvement. Although continuous improvement is a pivotal element of Scrum and other agile methodologies we are faced with the problem to identify potential for improvement. The identification of improvement potential should be done in an objective rather than subjective way. Software metrics can be used to fill this gap. A variety of software metrics exist that provide information about resources, processes and products involved in software development. Software metrics provide factual and quantitative information.

The introduction of software metrics alone is not sufficient. Many software measurement programs have failed because they don't respect important success factors [35]. Niessink and van Vliet state that "...for a measurement program to be successful within in its larger organizational context, it has to

generate value for the organization. This implies that attention should also be given to the proper mapping of some identifiable organizational problem onto the measurement program, and the translation back of measurement results to organizational actions" [35].

Thus, a measurement program has to reflect specific organizational problems or goals in order to deliver data which can be used to find solutions to the organizational problems that relate to the organizational goals. In other words, the measurement program will only able to deliver relevant information if a link between the business goals and the performance measurements is established.

But even if the measurement program reflects the goals of the organization, it is difficult to classify the measurement results. A person trying to interpret a software measurement is confronted with the question, whether a specific metric value is an indication of improvement or deterioration with respect to a certain objective.

All these problems need to be solved to successfully establish a software measurement program. The purpose of this thesis is to investigate ways to solve the above mentioned problems of software measurement. The research is specially targeted at an agile environment.

1.2 Research Questions

The problems stated in the previous section lead to the main research question of *how software measurements can be incorporated in an agile software development process to support continuous improvement*. The following research question were derived to find an answer to the main research question.

RQ1 : What approach should be followed to define a software measurement model in an agile environment?

RQ2 : What prerequisites must be fulfilled to ensure that the software measurement model delivers significant information?

RQ3 : What are the measurement goals in an agile software development process?

RQ4 : How can the metrics be effectively incorporated in an agile software development process?

RQ5 : How useful is the proposed software measurement model in an agile environment?

1.3 Research Context

AUTOonline was founded in 1996 to offer the first professional Internet-based platform to determine the salvage value of damaged vehicles as well as to buy and sell vehicles which were involved in an accident.

Vehicle assessors and insurance companies put damaged vehicles on the platform. Interested professional dealers submit their bid. The bids are placed hidden, which means that the dealers don't see the bids of other dealers. On the basis of the bids, an expert can determine the damaged car's salvage value. The expert opinion is sent to the vehicle owner who can then decide if he wants to sell the car to the bidder or not.

AUTOonline is not only the leading market place for vehicles involved in accidents in Europe but has obtained an important role in the advertisement of fleet vehicles as well. AUTOonline operates in over 28 countries and is part of Solera Holdings Inc. since October 2009.

Solera uses the principles *Think 80/20*, *Act 30/30* and *Live 90/10* to communicate the vision within the organization. The 80/20 principle sets the guideline that "...the focus should be on the 20 percent of the work that drives 80 percent of the value to our customers and employees" [6]. "The 30/30 principle emphasizes reducing waste and wasteful processes by 30 percent while improving throughput by the same amount" and the 90/10 principle prescribes to take 90 percent of the accountability and leave only 10 percent to others. These principles also form the basis for decision-making at AUTOonline.

The software products and services offered by AUTOonline are developed in an internal software development department. AUTOonline uses the .NET framework, the programming language C# and other state of the art technologies for software development. Rich client applications are developed using Windows Presentation Foundation and web applications are developed with ASP.NET.

The in-house software development enables AUTOonline to react to new or changed requirements in a timely manner and tailored to the specific needs of the different customer groups (vehicle assessors, garages, fleet marketers, buyers and insurance companies). To further improve the software development with a strong customer orientation, AUTOonline has introduced the agile software development methodology Scrum. During the introduction of Scrum the software development department was split into two teams. The teams have the responsibility for particular products and services of AUTOoonline. Each team consists of a product owner, a scrum Master and the software developers. The customer role is taken by sales representatives of AUTOonline. The sales representatives prioritize in cooperation with the

product owner the enhancements of the software products and services.

Since the introduction of Scrum a lot of experience has been gained with the new iterative development process. In order to further improve the process, AUTOonline wants to employ software performance measurements. Therefore, AUTOonline is faced with the same problems and question, as addressed in the problem statement. The research questions of this master thesis are aimed to find solutions to the problems. A measurement model will be established and validated in the business context of AUTOonline.

1.4 Research Method

The first step towards answering the main research question is to find a theoretical basis for the definition of a measurement model (RQ1). For this purpose, we study the literature. During the literature study we will also investigate how to choose and combine the metrics appropriately (RQ2). The implications from the first two research questions help to ensure that measurement models deliver significant data and that the data can effectively be used by the stakeholders involved in the software measurement.

RQ3 regarding the measurement goals in an agile software development process we will partly explore by means of the literature. The exploration tries to find out what industry experts define as the main goals in an agile software development process. The other part consists of the analysis of the AUTOonline business context. By looking at the implementation of the Scrum methodology, we will investigate what are the specific needs in practice. The analysis of the business context will be used to find answers to RQ4. We will analyze how exactly the development process is executed. What data is available? What data can be collected with respect to the implications of RQ2 and how can the information gained from the measurements be used effectively in the Scrum process? The results of the investigation of the first four research question constitute the basis to define a measurement model for the use in an agile environment.

Subsequently, we will apply the software measurement model to the business context. The application of the model will then be used to evaluate the software measurement model (RQ5). The evaluation contains the analysis of the usefulness of the software quality model and the collected data. For the purpose of the usefulness analysis, we will conduct interviews with the stakeholders of the measurement model.

1.5 Thesis Outline

Chapter 2 will provide background information relating to context of this master thesis. The common ground of the agile methodologies will be described. Following, an introduction is given to the Scrum methodology which represents the agile methodology that is employed at AUTOonline. Then we will discuss the term issue tracking system and how the issue tracking system is used at AUTOonline with special attention to the application in an agile environment.

Chapter 3 shows the approach which is followed to define a software measurement model for the use in an agile environment. Also an explanation of the prerequisites which must be adhered to when choosing metrics and combining them to a software measurement model. To support the interpretation of particular measurement scores, they are categorized using a ranking system. The method to calibrate the ranking system is also presented in Chapter 3.

Chapter 4 illustrates how the measurement goals for the software development department were derived from the business goals. Subsequently, the software measurement model is defined in this chapter. The presentation of the model contains all the metrics involved, as well as their relationship among one another.

Chapter 5 presents the results from the application of the proposed software measurement model at AUTOonline. Furthermore, the results of the validation from the stakeholder perspective are given.

Chapter 6 discusses the findings of the master thesis' research, recommendations and suggestions for future work are given.

Chapter 2

Background

This chapter gives information on which the study is founded. First of all, we give a summarizing description of the agile methodologies. Next, we discuss Scrum which is the agile methodology which is employed in the business context of our study. The last section of this chapter addresses the issue tracking system and how it is incorporated in the agile software development process.

2.1 Agile Methodologies

Traditional software development methodologies were deduced from engineering disciplines such as civil or mechanical engineering [20]. But the adopted concepts are not suitable for every software engineering project, because of the lack of flexibility.

The engineering disciplines have a clear separation between design and construction [20]. The separation can be made in construction because design is mainly an intellectual activity whereas the construction is foremost a physical activity. A comparable segregation is not possible in software engineering because a continuum exists between design and construction where both are thoroughly intellectual activities [42].

In civil engineering both phases are executed sequentially which means that once the design phase is finished it is not reentered. Projects which are strictly executed according to such a sequential process are predictable. Traditional software development methodologies try to plan out a large part of the software process in great detail for a long span of time [20]. The requirements are collected in the beginning of a project and then these requirement pass in one big bulk through all the steps in the development process (design, implementation and verification). But the same level of predictability

can't be achieved in software development as in construction because of the inconstancy of design.

The traditional software development methodology is the waterfall model. In this model UML diagrams are sometimes created as design documents. Although you can use peer review to check the design, errors in the design are often times only uncovered during coding and testing [20]. Furthermore, requirements often change even late in a project. These circumstances show that a high risk exists of the design becoming obsolete. Once the design must re-engineered a lot of the invested time becomes wasted effort. The high risk of wasted effort constitutes one reason why a detailed planning is very inefficient.

But there is another reason for the inefficiency of a detailed planning. Jim Highsmith states that “Projects may have a relatively clear mission, but the specific requirements can be volatile and evolving as customers and development teams alike explore the unknown” [24]. This fact adds to the risk of producing waste. Either time is wasted planning requirements or time is wasted on implementing requirements which the customer no longer needs. From the desire to cope with the unpredictability the agile software development methodologies emerged.

Agile methods are adaptive rather than predictive [20]. Agile methods are not only able to adapt to changes in the customer requirements but are also able to react to changes in the environment by adapting the software development process. Through these properties they are very effective in changing environments.

Several different agile methodologies exist. Scrum and Extreme Programming are only two examples but they all have the four guidelines of the *Agile Manifesto* in common [10]:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

The agile manifesto was created by the founders of the several agile software development methodologies, like Kent Beck (Extreme Programming), Ken Schwaber and Jeff Sutherland (Scrum). The guidelines result from the experience which they have gained with agile software development and are intended to improve software development.

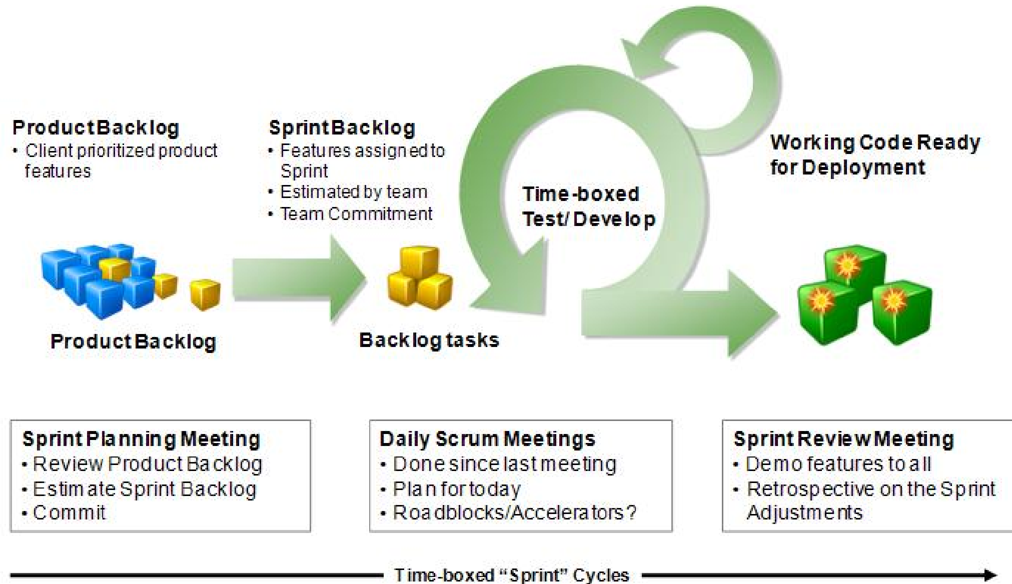


Figure 2.1: The Agile Scrum Process (Trustman and Davis[1] 2006)

The authors of the agile manifesto derived several principles from the four main guidelines. One of these principles is "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software" [11].

2.2 Scrum

Scrum is an agile software development framework which also has the four principles of the agile manifesto at its core. In a scrum environment a team develops software in sprints. The length of a sprint is consistent but different teams may use different sprint lengths. Usually sprint lengths of 2 to 4 weeks are used.

A sprint is an iteration which consists of several steps as illustrated in Figure 2.1. At the beginning of each sprint stands the sprint planning. In the sprint planning meeting the development team meets with the product owner who is responsible for the business value of a project. Together with the product owner the development team chooses a set of work items from the product backlog to implement during the sprint. The product backlog contains all the work items for a specific software product ordered by their priority.

Before the selection of the work items, a measure of complexity is assigned to each work item by the development team. The measure of complexity also known as story points is an estimation of the complexity to realize a particular work item. On the basis of the user story estimations and the statistic about the number of user story points completed in the last sprint (velocity), it is determined how many and which work items can be completed in the next sprint.

The selected work items constitute the sprint backlog. Once the team has committed to a sprint backlog the sprint starts. Every day the development team holds up a daily scrum meeting. In this short meeting it is discussed which progress has been achieved concerning the sprint backlog and if roadblocks exists which threatens the fulfillment of the sprint target.

At the end of each sprint all the completed work items are presented to the product owner. The product owner determines whether the work item are implemented according to the requirements. If the sprint was accepted by the product owner the product increment can be delivered/deployed.

The sprint is concluded by the sprint retrospective. Here the development teams looks back on the sprint and determines aspects of the product and process which need improvement. The scrum master is responsible for removing obstacles which were found during the sprint or in the retrospective. The regular feedback loop ensures constant improvement and a functional and productive development team.

2.3 Issue Tracking Systems

An *Issue Tracking System (ITS)* is a software tool which is used in the change management of software engineering projects. Issue tracking capabilities enable a project team to record and track the status of all outstanding issues [37]. In the context of this master thesis an issue is either of the type enhancement, bug or task. In addition to issue recording and status tracking of issues, ITSs are used for project planning.

Issue tracking systems, like Jira, offer extensions for agile software development. In the ITS context a work item is named *issue*. At AUTOOnline the issue in the ITS are either of the type *User story*, *Bug* or *Task*.

A user story describes a software enhancement from the user perspective. Usually, it contains the user requirements and the acceptance criteria which are used to evaluate whether an enhancement has been implemented according to the requirements. An issue of type bug is created if a deviation

[1] `http://www.1dot0.com`

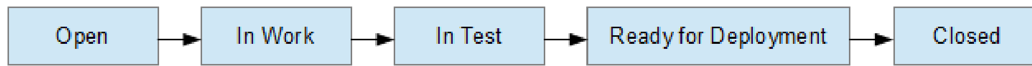


Figure 2.2: Issue Life Cycle

from the specified behavior is discovered in a software product or an error occurred during the execution of a software product. A task is a work item which doesn't influence the source code. An example of a task is a performance analysis of a software product or an investigation on new technology which could be used for the software product or process.

The different issue types go through the same life cycle states. The life cycle of an issue is shown in Figure 2.2.

If an issue is reported, it is linked to a product. At this point the issue has the *Open* state. All the issues in the *Open* state linked to a product constitute the product's backlog.

The issues which were selected in the sprint planning are transferred from the backlog to the sprint backlog. Once the sprint has started the software developers pick up issues from the sprint backlog and put them in the *In Work* state. On a daily basis, the progress of the issues in work is discussed and the plan for the day is presented. If the work on an issue is finished, the developer transfers the issue to the *In Testing* state. Issues in this state are picked up by the testing department. If the implementation of an issue successfully passes the test phase, the issue is moved to the *Ready for Deployment* state. After a product increment with the particular issue has been deployed on the production system the issue enters the *Closed* state.

ITSs often offer *burndown-charts*. Burndown charts are used in a scrum environment to display progress towards the sprint target. The sprint burndown chart relates the quantity of work remaining in story points (on the vertical axis) and the time elapsed since the start of the sprint (horizontal axis). In this context work remaining considers every issue which is on the sprint backlog and resides in the *Open*, *In Work* or *In Test* state.

Another variant of the burndown chart is the product burndown. Instead of putting the focus on the current sprint, it focuses on progress of the entire project.

Chapter 3

Constructing a Software Measurement Model

This chapter is focused on the research questions *RQ1: What approach should be followed to define a software measurement model in an agile environment?* and *RQ2: What prerequisites must be fulfilled to ensure that the software measurement delivers significant information?* We explain which aspects are important to pay attention to when implementing a software measurement program successfully. The rest of the chapter addresses some of the challenges connected to implementing a software measurement program. First, the approach is described which was employed to establish a software measurement program. Afterwards, the method is presented which is used to compare the performance of different sprints.

3.1 Prerequisites for Software Metrics

The choice and combination of metrics is important in order to support decision making. There are several things to consider when establishing a software measurement program. One pitfall is named *Metric in a Bubble* [14]. This situation arises when you look at a metric in isolation. You need to put the metric in a context in order to derive useful information from the collected data. The context can, e.g., be achieved by relating the most recent metric value of a software product to historical values of the same metric and software product. Only then you are able to see if the software product is improving or declining on the metric. On this basis you can then identify possible reason for the trend.

The most common pitfall, according to Bouwers et al., is what they call *Treating the Metric*. This describes a situation in which alterations in behav-

ior are solely done to score better on a metric and not to improve towards a goal, "At this point, the value of the metric has become a goal in itself, instead of a means of reaching a larger goal" [14]. In order to inhibit such a disadvantageous change in behavior, it is important that the people involved in the software measurement program understand how the software metrics relate to the measurement goals. This way the people being measured see the relevance of the measurement and they can use the results of the software measurement to adjust their behavior to the organizational benefit.

Moreover, not only the relationship between the metric and the measurement goal should be apparent but the nature of your goal should be accommodated in the choice of metric. Often times there are several aspects to a goal which need to be measured. In situations where some aspects are left out of considerations, you rather hinder than support decision making. For example, measuring the productivity of developers in lines of code and at the same time leaving out a measure for code quality promotes the treating of the metric. The reason is that the developer is inclined to focus on producing lines of code and disregards the importance of good design. Bouwers et al. refer to the one-dimensional representation of a multi-dimensional goal as the *One-Track Metric* [14].

The opposite of too few metrics is also obstructive. If you include too many metrics in your measurement program, it becomes impossible to assess the progress towards measurement goals. The improvement of one metric score always causes the decline in another metric score. The person who is being measured starts to reject the metrics and its goals altogether because he can't see how to improve on them. This is called *Metrics Galore* [14]. Another risk of metrics galore is that you introduce metrics which can't directly be influenced by the people responsible to reach the measurement goal. This threatens the acceptance of the measurement program, too.

All the metrics for your measurement program should be chosen with respect to the ease of collection and degree of validity. Metrics should be collected automatically as much as possible. Manual interactions required to collect measurements impedes frequent measurement execution and is counter-productive because the invested time goes to the expense of time available to produce value for the company. Additionally, the manual involvement in the measurement threatens the validity of the data. The validity of the measurement data is important because wrong or imprecise data can lead to wrong interpretations and eventually to wrong decisions. Hence, the uncertainty of a measurement must be considered when interpreting its result. Measurements with a very high uncertainty must be neglected because no valuable information can be derived from them.

For the purpose of supporting the improvement of the software devel-

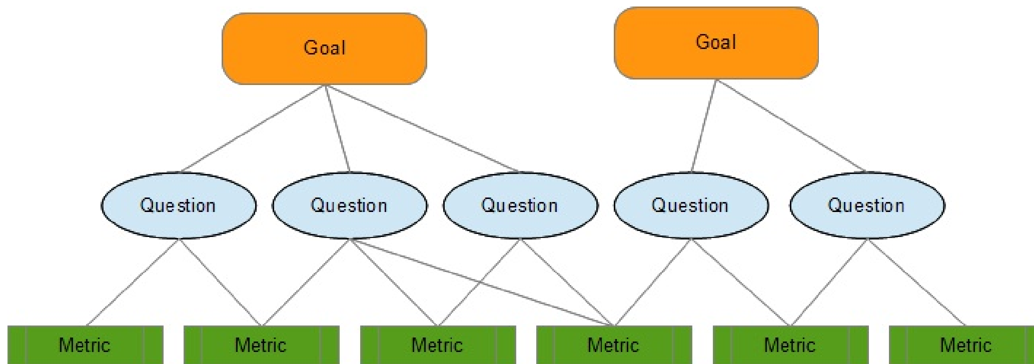


Figure 3.1: Goal Question Metric Approach

opment process and products, the metrics should enable root-cause analysis [23]. In other words, by looking at the measurements, it should become obvious which part of the process or product causes a problem concerning the achievement of the measurement goals. In Chapter 5 we will show how the software quality model can be used to identify potential problems. Once the cause of the problem is identified, actions can be planed to solve the problem and to improve the software development process.

3.2 Approach to Define a Software Measurement Model

During the long existence of software metrics, there has always been a strong debate if software metrics can fulfill the desired purpose. Fenton and Neil say that traditional metric approaches have failed in supporting decision making adequately [19]. As a reason for the failure, Fenton and Neil identified the lack of an explanatory framework [19].

The *Goal Question Metric (GQM)* approach which was defined by Basili is used in this study to define the measurement model and serves as an explanatory framework for the performance assessment. Basili says, “for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals” [8]. The GQM approach, illustrated in Figure 3.1, derives software measurements from specific performance goals. These performance goals are also referred to as *measurement goals*.

For each measurement goal, one or more questions are posed which char-

acterize how the goal can be achieved. In order to assess the achievements towards defined goals, we need to answer these questions. For this purpose, we select metrics which provide us with the information required to answer the questions. In other words, the metrics quantify a factor which influences the performance achievements towards our measurement goals. To support the interpretation of a specific metric, Basili advises to create an interpretation model and provide it to the people involved in the measurement program [9].

The top-down approach to define a measurement program combined with an interpretation model solves the problem described by the *Metric in a Buble*.

3.3 Linking Business Goals To Measurement Goals

Goldratt says, "Every action taken by any part of the organization - should be judged by its impact on the overall purpose. This immediately implies that, before you can deal with the improvement of any section of a system, you must first define the system's global goal; and determine the measurements that will enable us to judge the impact of any subsystem and any local decision, on this global goal" [32]. This same position is also taken by Basili, "This linkage [between software measurement goals and higher-level goals of the organization] is important, as it helps to justify software measurement efforts and allows measurement data to contribute to higher-level decisions" [9].

This master thesis considers measurement goals from the perspective of a profit-oriented business. Although the mission, values and strategies of profit-oriented business differs from company to company, they share the same overall purpose. The overall purpose of a profit-oriented business is expressed by the word *profit* in the categorization of the business. The mission, values and strategies determines how a particular company targets to achieve the overall purpose.

Net Profit (NP) and Return On Investment (ROI) are common measurements to assess the profitability of a company. But NP and ROI are not practical for managers in day-to-day operations. Traditionally, cost accounting is used by managers as a tool to support day-to-day decision-making. But Corbett states that cost accounting is obsolete, because it is based on erroneous assumptions [18].

According to Anderson the problem with cost accounting is that the focus

on cost efficiency leads to erroneous conclusions, "Cost accounting assumes that local efficiency leads to global efficiency. The problem with this is easily exposed. Because inactive men or machinery are not truly variable, the costs they incur are placed in a bucket known as 'overhead'. Overhead is cost assigned to the system as a whole. Hence, local efficiency is no indicator of global efficiency" [5].

Although the shortcomings of cost allocation might be fixable, Corbett goes further and questions the need to allocate cost to products in order to make good decisions [18]. He also criticizes that cost accounting emphasizes cost efficiency. Performance measurement leads to behavior adaption of the people being measured or, as Goldratt formulates it: "Tell me how you will measure me, and I will tell you how I will behave"[22]. Knowing this, it is not desirable to put the focus solely on cost efficiency because it is an one-dimensional representation of a multi-dimensional goal. Applying cost accounting therefore relates to the one-track metric concept which needs to be prevented. In order to prevent a one-track metric cost efficiency metrics need to be complemented with other metrics.

Corbett advocates the usage of Throughput Accounting (TA) instead of cost accounting. TA builds on the Theory of Constraints (TOC) which was proposed by Goldratt. Anderson states that "... TOC assumes that a value chain is only as strong as the weakest link in the chain. The capacity of the weakest link is the current system constraint" [5]. Hence, in order to improve the throughput of a system we need to focus our effort on the utilization and improvement of a system's constraint. TOC uses the measures Throughput (T), Operating Expense (OE) and Investment (I). These measures are defined as follows:

- Throughput: "is fresh money that has two sides, Revenue and the Totally Variable Costs (TVC) ... TVC is that amount incurred when one more product is sold. You have product and company's throughput. A product's throughput is its price minus its Totally Variable Cost. A product's contribution to the company's throughput is its throughput multiplied by the number of units sold." [18].
- Investment: "is the name given to the money tied up in the system." [39]
- Operating Expense: "is the name given to all the money (other than TVC) incurred to turn investment into sales this includes wages etc." [39]

On the basis of these three measurements, the NP and ROI can be calculated as followed:

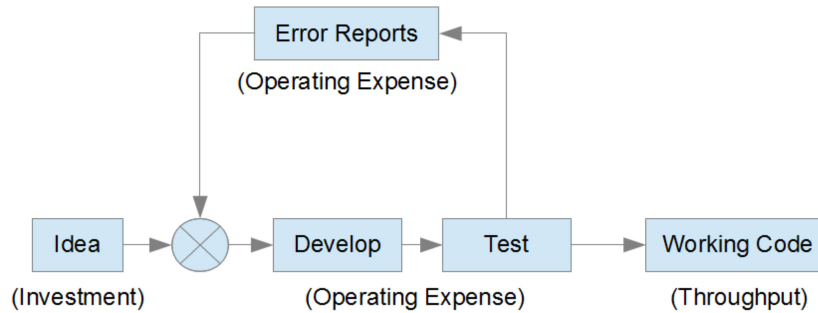


Figure 3.2: Throughput Accounting applied to a Software Development Process (Anderson 2004)

$$\text{Net Profit} = \text{Throughput} - \text{Operating Expense}$$

$$\text{Return On Investment} = \frac{\text{Net Profit}}{\text{Investment}}$$

By looking at the measurement formulas you can see that NP can be increased by increasing the Throughput or by decreasing the Operating Expense. The ROI can be increased by increasing the NP as described or by decreasing the Investment. The NP and ROI calculations from throughput accounting enable managers to assess the profitability of the company and moreover represent a starting point to plan actions for improvement.

The concept described by TA is a high level perspective on a company's performance and the challenge remains to apply this theory to lower organizational levels. Anderson applied throughput accounting to software development. For this purpose Anderson looks at a software development process and defined what throughput, operating expense and investment means in this context. Figure 3.2 shows a simplified model of a software development process which illustrates how a single idea/requirement passes through the system. Additionally, he defines which steps in the development process contribute to which measurement (Investment, Operating Expense or Throughput) from TA. The effort used to create an idea or receive a user requirement is considered as investment. The costs which accrue during the transformation steps *develop*, *test* and *error reports* make up the operating expense. Finally, throughput is the monetary value of the working code.

As Anderson shows, throughput accounting can be applied to the software development process to assist decision-making. But in practice it is often a challenge to obtain valid data to assign to I, OE and T. Especially, the assignment of a monetary value to working code is a problem. In software development we can calculate what throughput was generated by a

Figure 3.3: Levels of Decision Making

particular product or service for a given time period. Whereas we can't calculate what the implementation of a particular user requirement into working code contributes to the product's throughput. Therefore, the application of throughput accounting has its limitations.

Although throughput accounting is able to support decision-making under the mentioned conditions, it is not applicable to decision-making on all levels. Decision making takes place on three levels (strategic, tactical and operational). The Figure 3.3 conveys the hierarchical structure in which organizational goals and strategies relate. The strategies on the lower levels follow from goals and strategies which were defined on the levels above. Throughput accounting can be used to set goals on the strategic and tactical level. In the review of the past performance, throughput accounting can be used to assess if the employed strategies were successful to reach the predefined goals. But on the operational level throughput accounting can't be used to set goals and assess the achievements towards these goals.

In an agile environment a lot of decision-making is delegated to the self-organizing development teams. Development teams need information from software measurement as assistance in their day to day operations. At the organizational level the development teams are faced with planning, progress evaluation and retrospective of an iteration. Throughput accounting doesn't deliver the information required to support these tasks.

Hence, in order to be able to define measurement goals and metrics for the operational level, we need to analyze how software development contributes to the creation of business value.

Business value is an informal term. The value of a business is determined by its tangibles but also by its intangibles. Highsmith says that these intangibles are important for a company's long term success,"...intangibles are

critical to long-term success, and the ability to capitalize on the flow of opportunities is a critical intangible capability for most companies" [27]. The ability to capitalize on the flow of opportunities is connected to the *Agility* from a software development point of view. Agility is defined by Highsmith as "...the ability to both create and respond to change in order to prosper in a turbulent business environment. It means we have to build and sustain a continuous value delivery engine, not one that bangs out many features the first release and then rapidly declines in ability thereafter. The ultimate expression of agility from a software perspective is continuous delivery and deployment" [26]. Therefore, a high agility is key to business success. Disregarding the need to assess and improve the agility will lead to missed opportunities and consequently will result in financial loss. Hence, intangible factors like agility have an immense effect on tangibles. The software quality model focuses on the intangible components of business value.

Chappell sees software quality as an important intangible contributing to the business value, "Every organization builds custom software for the same reason: to create business value. An essential part of that value stems from the quality of the software"[15]. The consequences of low software quality over time are obstructive to the business. According to Chappell low software quality will lead to financial loss from lost business, financial loss from customer reparations, financial loss from lost customers, financial loss from lawsuits and eventually lower brand equity [15].

Hence, to guarantee that software development adds to the business value creation, we need to assess and improve the software quality. For this purpose we need to break the software quality aspects down into measurable components. Chappell specifies several factors having an influence on the software quality aspects (see Table 3.1).

Functional quality deals with factors which are particularly important for stakeholders with an external view on the software development. Stakeholders with an external view evaluate the *value* of a software product along the functional quality attributes. For example, a user analyzes if a software product offers the functionality which she/he needs. If the functionality doesn't adhere to the user's requirements, the value of the software is diminished to that customer. Consequently, the functional quality influences the adoption or rejection by users.

Process quality refers to the state of the development process. Compliance with release and budget plans is do to the quality of the software development process.

Structural quality is an internal view on the quality of a software product. Relevant for the internal view is the testability or maintainability of the software product's source code. Although structural quality is an internal

Table 3.1: Factors influencing Software Quality (Chappell)

	Factors
Functional Quality	1. Meeting specified requirements 2. Creating software with few defects 3. Good enough performance 4. Ease of learning and ease of use
Structural Quality	1. Code testability 2. Code maintainability 3. Code understandability 4. Code efficiency 5. Code security
Process Quality	1. Meeting delivery dates 2. Meeting budgets 3. A repeatable development process that reliably delivers quality software

view, it still is perceived on the outside. Bad structural quality may result in more defects in the software products or in longer development waiting times for enhancements because it is simply more difficult to make changes in the source code of a software product. Hence, structural quality has great influence on the functional and process quality.

Highsmith mentions similar factors in his *Agile Triangle* which contribute to the business value and determine the success of software development [25]. The agile triangle in Figure 3.4 has the dimension *Value*, *Quality* and *Constraints*. The goals of agile software development are defined along these three dimension. Highsmith states that the goals of software development are, "...producing value that delights customers, building in quality that speeds the development process and creates a viable platform for future enhancements, and delivering within constraints (which are scope, schedule, and cost)" [25]. The agile triangle especially emphasize the importance of the creation of value in the form of releasable software product. The emphasize is inline with the principles behind the agile manifesto, like "Working

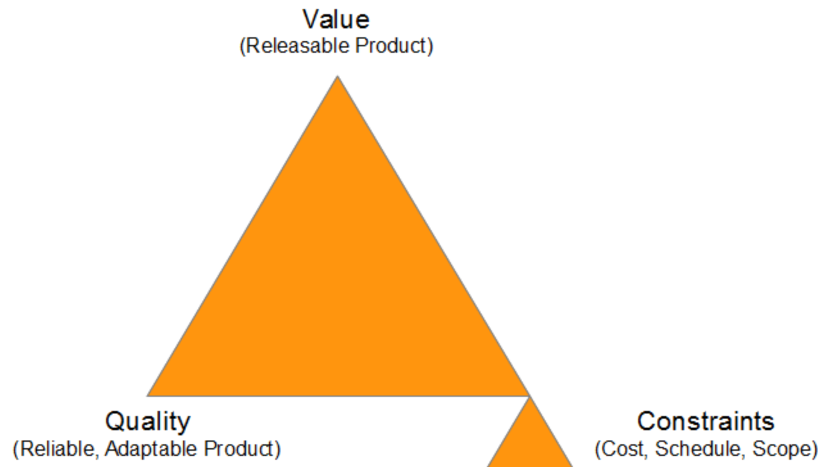


Figure 3.4: The Agile Triangle (Highsmith 2010)

software is the primary measure of progress" [11].

Following the reasoning of Highsmith and Chappell, we conclude that high software quality with a strong focus on the creation of value in form of working software is crucial to ensure that software development is contributing to business value creation. In other words, software quality is the key enabler to *Agility*. The higher the agility of software development, the higher is the ability of software development to support the capitalization of business opportunities and to contribute to the business value.

3.4 Method to Calibrate Metric Thresholds

Heitlager et al. use a rating system to rank the maintainability of software systems [23]. In the *SIG Maintainability Model* source code properties are mapped to maintainability characteristics. On the basis of the metrics relating to the source code properties, ratings are assigned to a software system representing the degree of maintainability. The ratings range from one star to five stars. One star meaning very bad maintainability and five stars very good maintainability. Such a ranking system enables us to measure system characteristics and moreover supports the interpretation of a metric. Alves et al. have continued the research of Heitlager et al. by proposing a method which derives thresholds from a benchmark to rank source code metric scores [4].

Using a benchmark is a way to create a context in which a metric score is interpreted. Putting a metric in a context prevents the principle of the metric in a bubble, as mentioned in the previous section. Although the

method is used to compare values of source code metrics between different software systems, it still can be applied to other type of metrics. The reason for this is the foundation on statistical analysis of the distribution and scale of a metric.

According to Alves et al., it is important that metrics thresholds, for example thresholds for the star ratings, are chosen so that they “bring out the metric’s variability” [4] and “help focus on a reasonable percentage of the source code volume” [4]. The latter quotation relates to source code metrics, but the principle also applies to other metrics. Generally, it is important to appreciate the things that went well during a sprint, but when it comes to continuous improvement you are especially interest in the things that didn’t go so well during a sprint. Hence, to be able to distinguish good from bad sprint performance, we must choose the metric threshold according to the metric’s variability. For example, a one star rating is a sign of very bad performance. The rating system should be calibrated so that only a low amount of metric scores get a one star rating. In the retrospective of a sprint we can analyze reasons for the one star ratings and thereby identify potential for future performance improvements. If we fail to choose the thresholds appropriately, too many scores will be rated with one star and the search after impediments for a better performance is hampered.

Given that this study is focused on agile software development environments, every metric will be calculated for a sprint (sprint level). This enables us to rate a specific sprint performance against the benchmark of all sprint performances. Consequently, the benchmark will consist of the past sprint performances. Some metrics in the software quality model are directly measured for the sprint level, like the enhancement rate. Others, like the cycle time, are calculated on the issue level and must be lifted to the sprint level. To arrive at a sprint rating for the cycle time metric, we must aggregate the cycle times of the user stories completed in the sprint.

In order to aggregate a metric to the sprint level, we first establish four risk categorize (low, moderate, high and very high) to classify a metric value. These categories are also derived from the analysis of the metric’s data distribution in the benchmark. Figure 3.5 shows the distribution of the cycle time metric in a quantile plot. As can be taken from the quantile plot, the cycle time metric follows power laws.

The most variability of the cycle time metric takes place in the tail of the distribution. Consequently, we need to choose quantiles from the tail of the distribution. In the case of the cycle time metric, we use the 60, 85 and 90 percentiles for the risk category thresholds. The thresholds were slightly adjusted to arrive at whole days. The resulting risk categories can be seen in Table 3.2.

Table 3.2: Risk categories for cycle time

Category	Thresholds
Low	0 - 7 days
Moderate	7 - 21 days
High	21 - 28 days
Very high	>28 days

The risk categories are used to create a quality profile for a sprint. The quality profile for the cycle time metric is created by calculating how many percent of the user stories fall in each category. For example, the risk profile for sprint 10 of team blue is $\langle 100.0, 42.11, 10.53, 10.53 \rangle$. 100.0% of all issues fall into the low risk category, 42.11% into the moderate risk category, 10.53% into the high risk category and 10.53% into the very high risk category. The percentages for the different categories are cumulative which means that, for example, for the percentage calculation of issues in the *High* category we count the number of issues in the *High* and *Very High* category and divide by the total amount of resolved issues in the sprint.

The risk profile of a sprint is then used to assign a rating between 1 and 5 stars (the more stars, the better). Table 3.3 shows the rating scheme for the cycle time metric which was derived from the benchmark.

Table 3.3: Rating scheme for cycle time

Rating	Moderate	High	Very high
⋆⋆⋆⋆⋆	25.0%	0.0%	0.0%
⋆⋆⋆⋆	46.7%	14.3%	11.1%
⋆⋆⋆	53.3%	20.0%	20.0%
⋆⋆	57.1%	40.0%	26.7%
⋆	-	-	-

Using this rating scheme, sprint 10 of team blue would receive a four star rating. A star rating is not fine-grained enough to adequately compare sprints. To improve comparability, linear interpolation is applied to receive a continuous scale between 0.5 and 5.5. After applying interpolation, the cycle time rating of team blue for sprint 10 is 3.55.

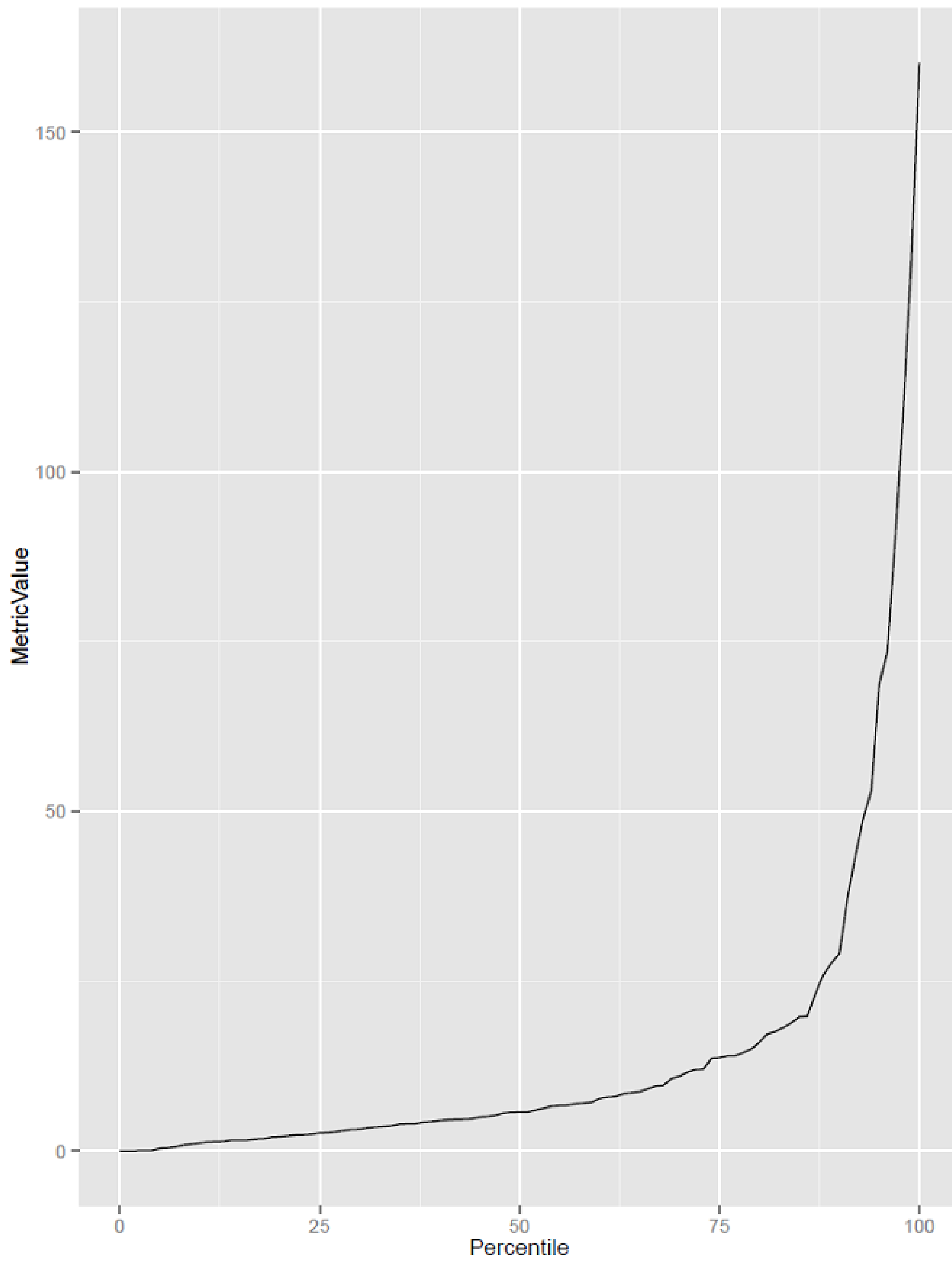


Figure 3.5: Distribution of Cycle Time

Chapter 4

Software Quality Model for Agile Environments

In this chapter we address RQ3. First, we will elaborate on what the main goal of software development is. It will be shown how the software quality model formed outgoing from the main goal using the GQM approach. We explain the sub-goals which were derived from the main goal and how they are refined with question until they are measurable with concrete software metrics.

4.1 Overview

The GQM approach, described in Section 3.2, is applied to define the model to assess the software quality. In Table 4.1 the main goal of agile software development is presented using the GQM definition template [40].

Table 4.1: Main Goal of Software Development

Analyze	Software Development
For the purpose of	Assessing and Improving Performance
With respect to	Software Quality
From the viewpoint of	Management, Product Owner, Scrum Master and Development Team
In the context of	Agile Environment

The analysis in the previous section already characterized the three aspects which contribute to the software quality. Figure 4.1 shows the GQM schema with *Software Quality Improvement* as the main goal and the derived sub-goals *Functional Quality Improvement*, *Structural Quality Improvement*

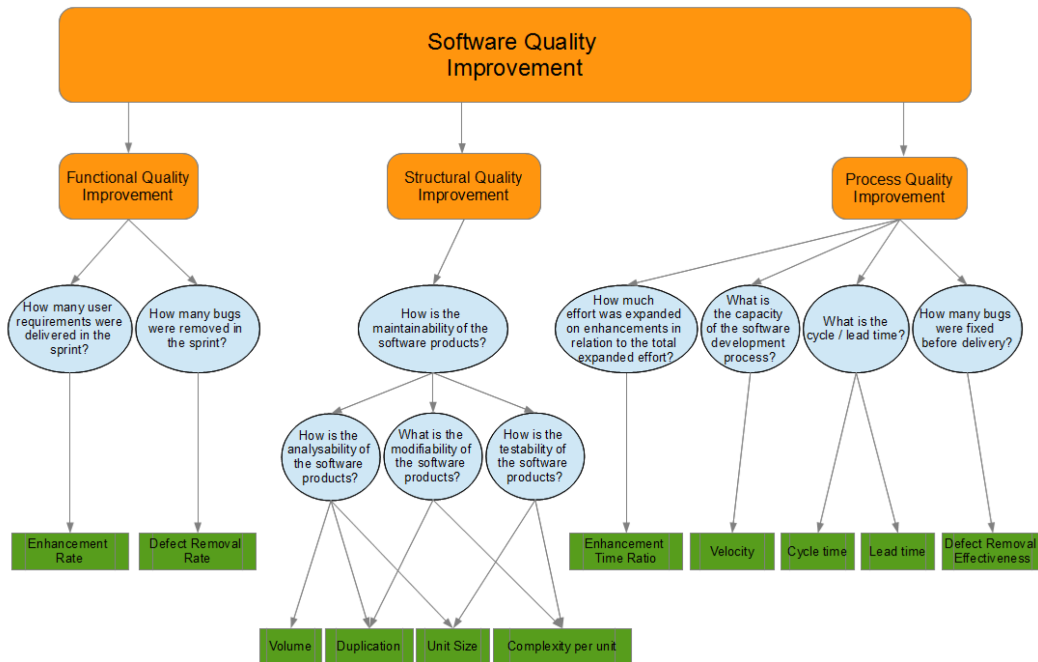


Figure 4.1: The application of the GQM approach to define the software quality measurement model

and *Process Quality Improvement*, as well as the questions and metrics which were derived from the sub-goals.

The different stakeholders in an agile development department put different emphasis on the characteristics of software quality. For example, a development team cares most about the structural quality, whereas the product owner cares most about the functional quality. It is important to note that a software development project often has additional external stakeholders, such as customer and/or user, which are not represented in the viewpoints of the model. These stakeholders are left out because they don't use this model, although it is important to incorporate their viewpoint in decision making.

The characteristics of software quality are interdependent. In certain situation a decision to improve on one characteristic may result in a temporary decrease in another characteristic. For example, the decision to refactor source code in order to improve the structural quality of a software product leads to reduced amount of features in this period. In an agile development environment, the development team has to make these trade-off decisions in cooperation with the product owner. The software quality model enables the team to assess the state of the software quality and thereby judge the effect

of their decisions.

Heitlager et al. [23] used a matrix to visualize the mapping of software maintainability characteristics onto source code properties. This method is adopted in this study to visualize the mapping of the software quality sub-goals onto product and process properties, as seen in Table 4.2.

		product and process properties							
		Enhancement Rate	Velocity	Lead Time	Cycle Time	Software Maintainability	Defect Removal Rate	Defect Removal Effectiveness	Enhancement Time Ratio
Software Quality	Functional Quality	x					x		
	Structural Quality					x			
	Process Quality		x	x	x			x	x

Table 4.2: Mapping of Software Quality Characteristics to Process and Product Properties

The software quality sub-goals are represented in the rows and process and product properties are represented in the columns of the matrix. A cross in the matrix symbolizes that the process and product property was determined to have a strong influence on the software quality sub-goal.

The characteristics functional quality and structural quality defined by Chappell can also be found in the ISO/IEC 25010, shown in Figure 4.2. Functional quality and structural quality of Chappell maps to the *Functional Suitability* or *Maintainability*, respectively. The functional suitability is defined as the “degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions” [28] and maintainability is defined as the “degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers” [28].

It is important to note that the other characteristics in the ISO 2510 product quality model, like *Performance Efficiency* and *Usability*, are of value. But the chosen characteristics are the most important to assess if software development is “producing value that delights customers, building in quality

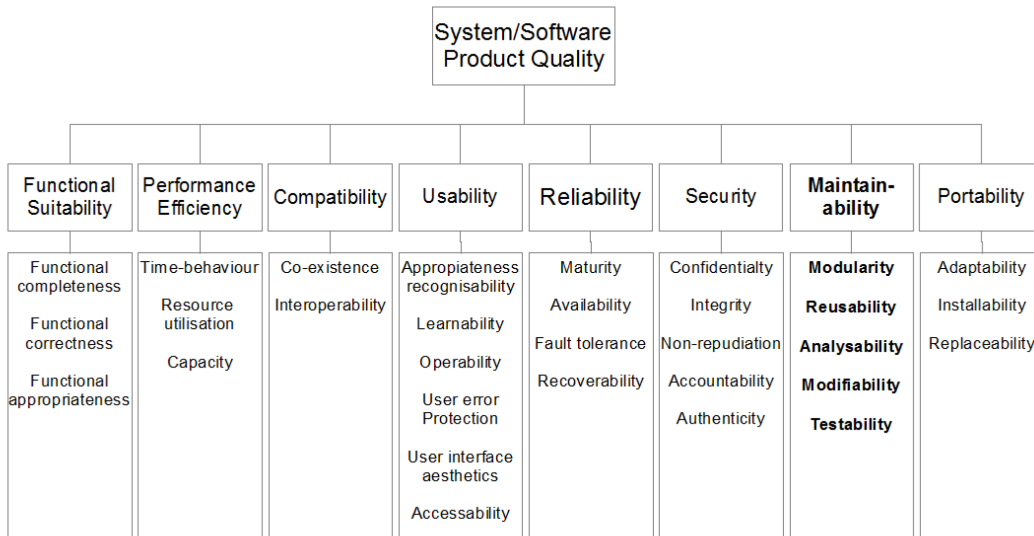


Figure 4.2: Product Quality model (ISO/IEC 25010)

that speeds the development process and creates a viable platform for future enhancements, and delivering within constraints (which are scope, schedule, and cost)" [25]. Another important fact is that the chosen characteristics can be assessed without much additional effort in an agile development environment, like AUTOonline. Most of the data for the assessment already exist in software tools, like an ITS which is commonly used in an agile development environment. Additional tools required for the analysis of a software product's maintainability can be integrated in the continuous build process of a software product. The continuous build is part of the *Continuous Integration* practice which is also commonly used in an agile development environment. This makes it possible to automate the data collection which greatly contributes to the efficiency of the software quality assessment.

If special requirements exist in a project which justify the additional effort required to assess further characteristics, like performance efficiency, the model can be extended to accommodate these requirements.

4.2 Goals

The GQM approach proposes to derive the metrics of a measurement from the goal of the measurement program. The main goal is to assess the software quality. The software quality characteristics *Functional Quality*, *Structural Quality* and *Process Quality* form the basis for the measurement program. Following the goals are described in more detail.

Functional Quality

The functional quality of a software deals with the quality perceived on the outside of a software. The higher the level of functional quality is the greater the value which the user receives from using a software product. The value which the customer receives from the software product determines if a person/company decides to use a certain product or service and sign a contract. Therefore, it is essential to promote a high level of functional quality to ensure the sale of your products and services.

Structural Quality

The structural quality, also referred to as technical quality, of a software is related to the structure of the software's source code. Hence, it is an internal view of a software's quality. Especially, the software developers care about this part of software quality because they are affected by the problems of low quality in this aspect [16].

Process Quality

The process quality considers aspects related to the delivery process. In order to assess the delivery process the adherence to delivery dates and budgets is analyzed. Chappell emphasizes that it is not sufficient to deliver features in time and on budget at one point in time, but you need to be able to consistently deliver features on time and in budget. The higher the process quality the higher is the ability to deliver features in time and on budget.

4.3 Questions

Following the GQM approach we formulated questions which help us to assess goals from the previous section. The defined questions are presented in this section.

Questions relating to Functional Quality

In order to assess the functional quality goal we use the factors of functional quality which were defined by Chappell. In section 3.1 we presented the obstructive influence of *Metrics Galore*. This situation occurs when a large amount of metrics overwhelms the stakeholders of a measurement program. Consequently, we concentrate on the most important factors *Meeting the specified requirements* and *Creating software that has few defects* which were

defined by Chappell. These factors relate to sub-characteristics of functional suitability in ISO 25010, shown in the following list:

- Functional completeness: "degree to which the set of functions covers all the specified tasks and user objectives" [28]
- Functional correctness: "degree to which a product or system provides the correct results with the needed degree of precision" [28]

The assessment if requirements are met is connected to the *Functional completeness*. The factor whether the software products have few defects is bound to the *Functional correctness*. As mentioned earlier, the model can be extended with questions and metrics concerning other functional quality factors like *Good enough performance* and *Ease of learning and ease of use* if this is deemed beneficial. These last two factors are also part of ISO 25010. They constitute the discrete characteristics *Performance efficiency* and *Usability*.

The questions derived from the functional quality goal are as follows:

Q1.1 *How many user requirements were delivered in the sprint?*

Context: The more user requirements are delivered to the customer, the higher is functional completeness. The benefit, which users receive from software usage, increases with the degree of the software's functional completeness. The delivery rate of user requirements is also considered to be the throughput of the software development process.

Metrics: Enhancement Rate

Interpretation: The higher the amount of user requirements that are implemented and delivered, the higher the functional quality.

Q1.2 *How many bugs were removed in the sprint?*

Context: A bug (or defect) is an error which occurs during the execution of a software. Every bug in a software degrades its functional correctness. The benefit, which users receive from software usage, is closely related to the functional correctness of the software [15]. Therefore, it is important that effort is expended to remove software bugs.

Metric:	Defect Removal Rate
Interpretation:	The higher the amount of defects that are fixed, the higher the improvement of the functional quality.

Questions related to Structural Quality

According to Chappell, the structural quality is related to the internal structure of a software. In order to assess the structural quality we analyze the maintainability of the software's source code. In ISO 25010 maintainability is defined as the "degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers". Already by the definition we can see that the software maintainability has influence on the other characteristics of software quality. ISO 25010 lists several sub-characteristics contributing to the software's maintainability. The most influential on the other software quality characteristics are

- Analysability: "degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified" [28]
- Modifiability: "degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality" [28]
- Testability: "degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met" [28]

The software quality will focus on the analysability, modifiability and testability in order to assess the maintainability of a software. The other aspects of software maintainability are left out for now to prevent the detrimental situation of *Metrics Galore*. But additional maintainability characteristics can be added to the model later on, if this is deemed appropriate.

The question derived from the structural quality goal is the following:

Q2 *How is the maintainability of the software products?*

Context:	Research has shown that the maintainability aspects of structural quality has a positive correlation with issue resolution times [13]. This finding

	relates to implementing new features, as well as fixing bugs. In other words, the higher the maintainability of a software the faster issues from the backlog can be resolved. From this fact, we can see that the structural quality has a great influence on functional and process quality because bugs fixes and features can be delivered quicker and the required effort is better predictable.
Metric:	SIG Software Maintainability
Interpretation:	The higher the maintainability of a software, the higher is the structural quality of the software.

Question related to Process Quality

The assessment of the process quality goal is concentrated on the *Meeting delivery dates* aspect of Chappell's definitions. The assessment of the *Meeting budgets* is not applicable in the proposed measurement model because of the focus on the sprint level. Usually, you don't have budget plans for individual features on this level. Nevertheless, the metrics which are defined to assess the meeting of delivery dates, can be used to assess the efficiency of software development concerning the expended developer effort. By improving the efficiency, more features can be implemented with the same amount of developer effort.

The aspect *A repeatable development process that reliably delivers quality software* will in part be accommodated by the combination and formulation of the metrics. For the assurance that the development process consistently delivers software enhancements with good quality, we emphasize the critical role of how the information gained from the model is used for decision-making. If you rigorously aim to increase the amount of delivered features, you will most likely disregard the structural quality of the software. Although you may observe an improvement in the rate of new features on the short term, you will lose the ability to adopt to environment changes in the long term. The reason being, that the maintainability of a software is negatively correlated with the required developer effort to make changes to a software [13]. Therefore, it is important to be aware of the trade offs you make along the software quality aspects.

The questions derived from the process quality goal are as follows:

Q3.1 *What is the capacity of the software development process?*

Context:	The capacity is defined as "The maximum rate of output of a process, measured in units of output

per unit of time" [17]. The capacity of a software development process relates to the rate in which user requirements are implemented.

Metric: Velocity

Interpretation: The higher the capacity of the software development process, the higher is the process quality.

Q3.2 *What is the lead time?*

Context: The lead time originates in *Lean Manufacturing*, where it measures the time elapsed between order placement and delivery. If we transfer the lead time concept to the agile software development, it describes the time between the identification of requirement until the requirement is implemented and delivered. The lead time is what the customer sees[30].

Metric: Lead Time

Interpretation: The lower the lead time of the software development process, the higher is the process quality.

Q3.3 *What is the cycle time?*

Context: The cycle time is related to the lead time. It measures the time between identification of an requirement and the time the requirement is implemented and ready for delivery. Therefore, it measures the requirement implementation process from a developer point of view. Often times, an implemented requirement isn't directly delivered after its completion. In an Scrum development process the product owner decides when an implemented requirement is delivered. In combination with the lead time, you can analyze where problems in the delivery process exist. You can distinguish if the time elapsed to implement a requirement or the time elapsed to provide the implementation to the customer is a problem.

Metric: Cycle Time

Interpretation: The lower the cycle time of the software development process, the higher is the process quality

Q3.4 *How much effort was expended on enhancements in relation to the total expanded effort?*

Context: During the sprint planning the product owner and the development team must plan which issues are going to be handled in the sprint and how the available developer time is expanded on software enhancements, bug fixes and tasks. It is important to ensure a constant stream of software enhancements to satisfy the user requirements. But at the same time it is also important to fix bugs to reestablish the intended benefits for software users and spend time on task, like the analysis of new software technology, to find potential improvements for the software products and process. Hence, it is important to balance the expanded effort on the three different issue types with a priority on the constant stream of software enhancements.

Metric: Enhancement Time Ratio

Interpretation: The higher the enhancement time ratio, the higher is the process quality.

Q4.4 *How many bugs were fixed before delivery?*

Context: Software development is done by knowledge workers and not machines. Although developers don't intend to introduce bugs to the software it is inevitable that they do. As mentioned earlier, the functional correctness and benefit which a user receives from a software is related to the number of bugs in the software. Knowing this, it is important that during quality assurance activities as many bugs as possible are identify and fixed before the software is delivered.

Metric: Defect Removal Effectiveness

Interpretation: The higher the amount of bugs that are detected and fixed before the delivery of the software, the higher the improvement of the functional quality.

4.4 Metrics

The metrics that were selected as indicators of software quality are defined and described in the following.

Enhancement Rate

The *Enhancement rate* is a measure of process throughput. The throughput of a process is the amount of units which flow through the process in a given time period. In software development we consider user requirements as input to the development process. A user requirement (idea) passes through the different stages of the software development process, as depicted in Figure 3.2. The output of the development process is the implementation of an user requirement in form of working code. The implementation of a user requirement constitutes a software feature or enhancement.

TA emphasizes that the output of the production process doesn't add value to the business until it is sold. We adapt this notion of TA to software development. Consequently, we only consider enhancements which are delivered to the customer/user to calculate the enhancement rate metric. This is also in line with the principles behind the agile manifesto, e.g. "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software" [11] and "Working software is the primary measure of progress" [11].

In an agile development the focus lies on the development iteration or sprint, as it is called in the Scrum methodology. Planning is executed for the length of a sprint and at the end of every sprint a retrospective is conducted with the intention to find improvement potentials for the development process. The chosen metrics are intended to support the analysis of potentials for performance improvements. For this reason, we use the length of a sprint as the time period for which we measure the amount of delivered enhancement.

In the ITS enhancements or user stories, as they are called in this context, are allocated to a specific sprint and the resolution state is also accessible for each user story. Hence, all the information required for this metric can easily be retrieved from the ITS. The enhancement rate metric is defined as follows:

$$\text{Enhancement Rate} = \frac{\#\text{ User stories resolved in sprint}}{\#\text{ Working days in sprint}}$$

Defect Removal Rate

Software developers strive to reduce the injection of defects during development, but they can't completely eliminate defects in software products. Defects present in a software diminish the value gained from using it. Therefore, it is important to remove as many defects as possible.

The *Defect removal rate* metric is used to analyze the amount of software defects which were successfully removed from the software products in particular sprint. Only bugs which were found/created before the sprint are counted against this metric. This calculation closely connects to the *Defect Removal Effectiveness*.

The data for the metric calculation is taken from the ITS, as well. Defects are marked in the ITS with the issues type *Bug* and are assigned to the particular sprint in which they are fixed. The following formula is taken to distinguish which bugs were found before the sprint:

$$\text{Bug found before sprint} = \text{Bugs with creation date} < \text{Sprint Start Date}$$

It is important to note that the naming doesn't convey that in this case only bugs are considered which were found before the sprint and are resolved in the sprint.

On the basis of the *Bug found before sprint*, the formula for the defect removal rate looks as the following:

$$\text{Defect Removal Rate} = \frac{\#\text{ Bugs found before sprint}}{\#\text{ Working days in sprint}}$$

Defect Removal Effectiveness

The Defect Removal Efficiency (DRE) was proposed by Jones to assess the effectiveness[1] of the quality assurance activities during software development [29]. The DRE metric calculates the percentage of defects which were found and fixed during development before the software is delivered. Jones claims that the level of defect removal efficiency has a great influence on customer satisfaction, "...high levels of customer satisfaction strongly correlate with high levels of defect-removal efficiency. Conversely, software firms whose defect-removal efficiency levels sag below 85 percent almost never have really happy clients because their software is too unreliable" [29]. DRE is defined as follows:

[1] Contrary to the name defect removal efficiency, the metric measures the effectiveness instead of the efficiency because the metric doesn't include a measure of expended effort or cost.

$$\text{DRE} = \frac{\text{Development defects}}{\text{Total defects}}$$

where *Development defects* is the number of defects found and fixed during a development iteration before the software is delivered to the customer and *Total defects* is the sum of development defects and the number of defects which were found by user or clients after the software has been delivered.

The *Defect removal effectiveness* metric is based on the DRE metric. As mentioned earlier, we concentrate on the evaluation of the sprint performance. Consequently, we take the sprint as the time period for the metric calculation. Comparable to DRE, we want to distinguish the defects which were found during development from the defects which were found after delivery. For this purpose we once again look at the ITS data. In the ITS defects have the issue type *Bug*. Defects which were found in the sprint or during development, respectively, can be identified by relating the creation date of the issue to the sprint start date. The resulting formula is as follows:

$$\text{Bug found and fixed in sprint} = \text{Bugs with creation date} > \text{Sprint Start Date}$$

The defect removal effectiveness is then calculated by relating the amount of defects found in the sprint to the total amount of bugs which were fixed in the sprint. The formula looks as follows:

$$\text{DRE} = \frac{\#\text{Bugs found and fixed in sprint}}{\text{Total}\#\text{bugs fixed in sprint}}$$

Enhancement Time Ratio

Fixing software defects is a form of corrective maintenance. As mentioned in the description of the defect removal rate, it is desirable to have as little defects in your software as possible. But of course, there is a down-side to spending effort in bug fixing which is that less time can be spent on perfective maintenance in form of software enhancement. Bijlsma has proposed the *Enhancement ratio* metric to monitor the balance between corrective and perfective maintenance. The enhancement ratio is defined as the following:

$$\text{Enhancement Ratio} = \frac{RE}{RD + RE}$$

where RE is the number of resolved enhancements for a certain time period, and RD is the number of resolved defects for that same time period [12]. The data for the metric calculation can easily be retrieved from the ITS for each sprint, as we did for the previous metrics.

Bijlsma confirmed in his study the hypothesis that "Developers of software systems with higher maintainability will implement more enhancements (relative to their total effort), compared to developers of systems of lower maintainability" [12].

The *Enhancement Time Ratio* is an extension of the original enhancement ratio metric. The enhancement time ratio metric considers the time effort which was expanded by the development on the particular issues. Furthermore, the metric considers *User story*, *Bug* and *Task*. In the enhancement ratio metric only issues of type *User story* and *Bug* are considered. The adjustments enable us to better analyze how developer time was expanded during the sprint. The enhancement time ratio is calculated as follows:

$$\text{Enhancement Time Ratio} = \frac{\text{Logged time on user stories in sprint}}{\text{Logged time on user stories, bugs and tasks}}$$

Software Maintainability

Heitlager et al. have developed the *SIG Quality Model* to assess the maintainability of a software product [23]. This model uses the definition of maintainability in the ISO/IEC 9126 to assign a maintainability rating to software product. Later the model was adjusted when ISO/IEC 9126 was replaced by ISO/IEC 25010. The ISO/IEC 25010 breaks down the product quality into eight characteristics, as shown in Figure 4.2.

In ISO 9126 the maintainability characteristic had the sub-characteristics *Analysability*, *Changeability*, *Stability* and *Testability*. The SIG quality model defines source code properties which are map to the sub-characteristics of maintainability. The following list shows the source code properties which are mapped to the sub-characteristics of ISO 9126 in the original quality model [23]:

Volume: The overall volume of the source code influences the analysability of the system.

Complexity per unit: The complexity of source code units influences the systems changeability and its testability.

Duplication: The degree of source code duplication (also called code cloning) influences analysability and changeability.

Unit size: The size of units influences their analysability and testability and therefore of the system as a whole.

Unit testing: The degree of unit testing influences the analysability, stability, and testability of the system.

The source code properties can be measured through source code metrics. For example, the *Lines of code (LOC)* is used to measure the volume and the *McCabe Cyclometic Complexity* is used to measure unit complexity. Using the method to rate metrics, described in Section 3.4, a rating is determined for the different source code properties. The ratings for the code metrics are derived as described in Section 3.4. The code metrics ratings are then averaged following the maintainability mapping in Figure 4.3 to arrive at a score for the maintainability sub-characteristics.

Maintainability sub-characteristics	source code properties: Volume	Complexity per unit	Duplication	Unit size
analysability	x		x	x
modifiability		x	x	
testability		x		x

Table 4.3: Mapping of Maintainability Sub-characteristics to Source Code Properties

The mapping of system characteristics onto source code properties is accomplished using ISO 25010. The source code mappings to the sub-characteristics are chosen using the definitions of *analysability*, *testability* and *modifiability* as mentioned in Section 4.3.

The scores for the maintainability sub-characteristics are aggregated using averaging to arrive at an overall maintainability rating for a software product. Finally, all the maintainability ratings for the software products of a development team are averaged again to arrive at an overall maintainability score for the team's software products.

Velocity

In a Scrum environment, story points are assigned to requirements which represents the effort required to implement the particular requirement. For the

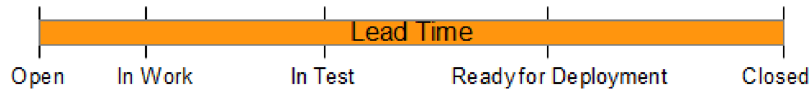


Figure 4.3: Lead time of an issue

capacity of the software development process from the development team's perspective we consider story points as the unit of output. As the time unit we consider the sprint length in working days. The required data for the metric calculation is taken from the ITS and looks as follows:

$$\text{Velocity} = \frac{\#\text{ Story points of resolved user stories in sprint}}{\#\text{ Working days in sprint}}$$

This way of measuring the velocity simplifies the sprint planning. Sometimes sprints vary in length, for example because of statutory holidays. Sprint planning is simplified because you can easily distinguish the amount of user story points which a development can implement during a sprint by multiplying the velocity metric with the amount of working days in a sprint. Moreover, is the comparability between teams for this metric improved because the metric accounts for the different sprint lengths.

Lead Time

The *Lead time* measurement originates in *Kanban*. Kanban is a method to control the flow of production. It was introduced at the Toyota Motor Corporation to improve the efficiency of the production process. The Kanban concept was adjusted by Anderson for the use in software development [31].

Anderson says, "The lead time from input to output will show how long Investment was committed within in the system" [5]. Accordingly, we define the lead time in software development as the time period beginning with the identification of an user requirement and ending with the fulfillment of the requirement in form of working code. In the ITS context, user requirements are have the issue type *User Story*. The identification of an user requirement is signified by the creation date of the user story or in other words the point in time when it enters the *Open* state. The point in time when the user story enters the *Close* state constitutes the fulfillment of the user requirement. Hence, the lead time covers the whole life cycle of an issue, as illustrated in Figure 4.3.

Each user story has story points assigned to it which represents an estimation of the effort required to implement the particular user story. A user story with more story points takes longer than a user story with fewer story points. We account for the effort in the lead time metric, in order to

Figure 4.4: Cycle time of an issue

achieve a better comparability between user stories. Therefore, the lead time is calculated as follows:

$$\text{Lead Time} = \frac{\text{User story closed date} - \text{User story open date}}{\#\ \text{Story points of user story}}$$

Cycle Time

In contrast to the lead time, *cycle time* refers only to a part of the whole life cycle. As depicted in Figure 4.4, cycle time covers the time period in the life cycle of a user story from the *Open* state until the user story enters the *Ready for Deployment* state. The *Ready for Deployment* signifies the point in time when the implementation of the issue is finished and all tests by the testing department have been passed successfully. The development team is only accountable for this particular time period because the decision if a product increment is delivered/deployed lies outside his responsibility.

Ideally, you should use the point in time when the developer starts his work on a requirement for the cycle time calculation. But this way of measuring would lead to undesirable behavior adoption in practice. If the cycle time starts with the *In Work* date, developers are inclined to delay the transfer of the ITS issue in order to receive better metric values. This obstructs the analysis of progress towards the sprint target.

Just as for the lead time, we use the story points for normalization to gain comparability among user stories. Therefore, the formula for cycle calculation looks as follows:

$$\text{Cycle time} = \frac{\text{User story ready for deployment date} - \text{User story open date}}{\#\ \text{Story points of user story}}$$

Chapter 5

Evaluation of the Software Quality Model

The software quality model was evaluated in a business context which is constituted by AUTOonline. This chapter presents the results of the evaluation of the software quality model. The first part of this chapter addresses the calibration of the metrics in the software quality model. Subsequently, we will analyze how the collected can be just to gain insight about the software development performance. Finally, we will discuss the results of the evaluation from the user's point of view. This chapter is aimed to provide answer for RQ4 and RQ5.

5.1 Model Calibration

AUTOonline implemented the Scrum methodology in the software development process. The performance of two development teams is going to be assessed using the software quality model. The teams are responsible for the maintenance of different software products and services. Each team plans and executes their sprint separately. The team sizes are comparable concerning the amount of developers in each team. However, the sprint length differs between the development teams. This fact was taken into account during the definition of the metrics in the software quality model.

The rating system for metric scores, which was presented in Section 3.4, is used during the application of the software quality model. For this purpose, a benchmark is established consisting of the data collected at AUTOonline. The data consists of the metric scores from each team over 6 sprints.

In Section 3.4 we mentioned that all metrics are aggregated to the sprint level using the rating system. Some of the metrics are already calculated on

the sprint level. The thresholds for these metrics are shown in Table 5.1. For all of these metrics applies that the higher the metric score the better the rating.

Table 5.1: Rating Thresholds for Sprint Level Metrics

Metric	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆
Enhancement Rate	2.38	1.64	1	0.87	-
Defect Removal Rate	1.38	0.53	0.29	0.13	-
Velocity	8.57	5.63	4.47	2.69	-
Enhancement Time Ratio	69.0%	65.0%	39.0%	33.0%	-
Defect Removal Effectiveness	85.7%	70.6%	60.0%	42.1%	-

The thresholds are interpreted as the minimum value to receive a certain star rating. An example for this type of metrics is the *Enhancement Rate* which measures the number of completed user stories in relation to the sprint length or workings days in the sprint, respectively. A software development team which delivered in average one user story per working day in the sprint will receive three star rating for this metric. Whereas, a team which only delivered in average 0.9 user stories per working day in the sprint will only get a two star rating.

Other metrics, like the *Cycle Time*, are measured for the issue level and need to be aggregated to the sprint level. For this purpose, we use the risk categories and the quality profiles to arrive at a rating for the sprint level. Table 5.2a shows the thresholds for the cycle time risk categories and Table 5.2b shows the quality profiles which were derived using the calibration method described in Section 3.4. Note that the lower bound of the *Moderate* and *High* category are not included. For example, a user story with a cycle time of 21 days will fall into the *Moderate* category and not the *High* category. The categories are then used to create the quality profiles. The quality profiles in Table 5.2b are also obtained using the calibration method.

The same calibration method was applied to receive the risk category thresholds for lead time which are shown in Table 5.2c. The associated quality profiles which are used to assign a lead time rating for a sprint are documented in Table 5.2d.

For the software maintainability we assess the maintainability of a team's software products at the end of a sprint. The software maintainability model has metrics which are measured for the software as a whole (system level). For example, the percentage of duplicated source code lines is calculated in relation to the software's total number of source code line. For the code metrics, which are measured for the system level, we derived the rating threshold

Table 5.2: Calibration Result for Cycle Time and Lead Time

Category	**Thresholds**
Low	0 - 7 days
Moderate	7 - 21 days
High	21 - 28 days
Very High	> 28 days

(a) Risk Category Thresholds

Rating	**Moderate**	**High**	**Very High**
⋆⋆⋆⋆⋆	25.0	0.0	0.0
⋆⋆⋆⋆	46.7	14.3	11.1
⋆⋆⋆	53.3	20.0	20.0
⋆⋆	57.1	40.0	26.7

(b) Quality Profiles Thresholds

Category	**Thresholds**
Low	0 - 14 days
Moderate	14 - 28 days
High	28 - 35 days
Very high	¿ 32 days

(c) Risk Category Thresholds

Rating	**Moderate**	**High**	**Very High**
⋆⋆⋆⋆⋆	0.0	0.0	0.0
⋆⋆⋆⋆	36.0	16.7	14.3
⋆⋆⋆	42.9	20.0	20.0
⋆⋆	46.7	40.0	22.2

(d) Quality Profiles Thresholds

presented in Table 5.3.

The unit size and complexity metric are measured on the unit level. In this case we consider the *method* code construct as the unit. The unit size metric counts the number of statements in a method. The risk category thresholds for the unit size metric can be found in Table 5.4a. The complexity metric measures the number of execution paths through a method. The calibration provided us the complexity risk category which are presented in Table 5.5a. The usage of quality profiles help us to aggregate the unit level code metrics to the system level. The quality profiles for unit size (Table 5.4b) and complexity (Table 5.5b) tell us how many percent of the source code may lie in the various risk categories to obtain a certain rating.

Table 5.3: Rating Thresholds for System Level Code Metrics

Metric	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	⋆⋆⋆	⋆⋆	⋆
Volume	3453	6329	39644	75001	-
Duplication	0.01	0.025	0.044	0.144	-

Table 5.4: Calibration Result for Unit size

Category	**Thresholds**
Low	0 - 20 statements
Moderate	20 - 40 statements
High	40 - 110 statements
Very high	110 statements or more

(a) Risk Category Thresholds

Rating	**Moderate**	**High**	**Very High**
⋆⋆⋆⋆⋆	0.0	0.0	0.0
⋆⋆⋆⋆	26.2	14.6	2.7
⋆⋆⋆	32.6	18.8	4.8
⋆⋆	75.2	65.6	51.3

(b) Quality Profiles Thresholds

Table 5.5: Calibration Result for Unit Complexity

Category	**Thresholds**
Low	1 - 5 complexity
Moderate	5 - 10 complexity
High	10 - 35 complexity
Very high	35 complexity or more

(a) Risk Category Thresholds

Rating	**Moderate**	**High**	**Very High**
⋆⋆⋆⋆⋆	1.9	0.0	0.0
⋆⋆⋆⋆	24.0	11.5	2.9
⋆⋆⋆	34.1	24.1	7.1
⋆⋆	72.8	64.8	53.3

(b) Quality Profiles Thresholds

5.2 Model Application

This section shows how the what information can be gained from the software quality model and how it can be used to assess the software development performance.

5.2.1 Evaluation of Sprint Performance

One purpose of this model is to assess the performance of a sprint. After every sprint the development together in cooperation with scrum master takes a look back onto the sprint in the sprint retrospective. In this meeting they try to identify things that impedes the software development performance. The metrics from the software quality model support the analysis for sources of improvements and can also be used to assess if decisions in prior retrospectives had a positive effect on the development performance.

The starting point of the sprint analysis is the software quality overview, as depicted in Table 5.6. In the overview you see the rating for the software quality together with the ratings of the software quality sub-characteristics.

Table 5.6: Software Quality for Team Blue (Sprint 10)

Metric Name	**Rating**	**Rating (Interpolated)**
Functional Quality	⋆⋆⋆	3.22
Structural Quality	⋆⋆⋆	2.51
Process Quality	⋆⋆⋆⋆	3.56
Software Quality	⋆⋆⋆	3.1

If the team wants to inspect a certain characteristic in more detail, they can drill down to view the concrete metric values for the characteristic. The Table 5.7 shows the scores of functional quality metrics of *Team Blue* for sprint 10. The team's performance on the functional quality dimension was average in sprint 10 compared to all sprints in the benchmark. It is noticeable that the rate of new enhancements, which were delivered in this sprint, was above average. The rating for the defect removal rate pulls the functional quality rating down but it is still a solid score compared to the other sprints.

Table 5.8 breaks down the process quality along its related metric scores. The overall process quality of the team was average for sprint 10. It is noticeable that the lead time and cycle time are on the very low end of three star rating. At this point the team may consider to drill down even further and look at individual issues to find reason which explain the low score. For this purpose, they can concentrate on the issues which fall into the *High* and *Very*

Table 5.7: Functional Quality for Team Blue (Sprint 10)

Metric Name	**Score**	**Rating**	**Interpolated Rating**
Enhancement Rate	1.73	⋆⋆⋆⋆	3.62
Defect Removal Rate	0.36	⋆⋆⋆	2.81
Functional Quality	-	⋆⋆⋆	3.22

High risk category of the two metrics. If the team find impediments which deferred the implementation or deployment of these issues, they can plan steps to resolve the impediment and to improve future sprint performances.

Table 5.8: Process Quality for Team Blue (Sprint 10)

Metric Name	**Score**	**Rating**	**Rating (Interpolated)**
Velocity	5.55	⋆⋆⋆	3.27
Lead Time	-	⋆⋆⋆⋆	3.62
Cycle Time	-	⋆⋆⋆⋆	3.55
Enhancement Time Ratio	67.0%	⋆⋆⋆⋆	4.0
Defect Removal Effectiveness	69.23%	⋆⋆⋆	3.37
Process Quality	-	⋆⋆⋆⋆	3.56

Table 5.10 shows the overall structural quality rating which is aggregated by averaging the maintainability ratings of the team's software project. Several projects only receive a two star rating. These low scores could be an explanation why some of the issues took a longer time to implement, if we can trace the issues back to these software projects. If low maintainability is indeed an impediment to the sprint performance, they can plan to invest time into refecatoring the project to increase the maintainability and thereby improving future performance.

Table 5.9: Structural Quality for Team Blue (Sprint 10)

Software product	**Rating**	**Rating (Interpolated)**
Project 1	⋆⋆⋆	3.27
Project 2	⋆⋆	2.16
Project 3	⋆⋆	2.06
Project 4	⋆⋆	2.08
Project 5	⋆⋆⋆	3.27
Project 6	⋆⋆	2.23
Structural Quality	⋆⋆⋆	2.51

Table 5.10: Structural Quality for Team Blue (Sprint 10)

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆	2.46
Duplication	⋆⋆⋆	3.18
Unit Size	⋆⋆⋆⋆	3.51
Unit Complexity	⋆⋆⋆	3.42

(a) Structural Quality of Project 1

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆	1.54
Duplication	⋆⋆	1.81
Unit Size	⋆⋆	2.28
Unit Complexity	⋆⋆⋆	2.54

(b) Structural Quality of Project 2

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆⋆⋆	3.98
Duplication	⋆⋆⋆	2.55
Unit Size	⋆	1.48
Unit Complexity	⋆	1.49

(c) Structural Quality of Project 3

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆	2.41
Duplication	⋆⋆	2.38
Unit Size	⋆⋆	1.93
Unit Complexity	⋆⋆	1.83

(d) Structural Quality of Project 4

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆⋆	3.47
Duplication	⋆⋆⋆⋆⋆	4.5
Unit Size	⋆⋆	2.68
Unit Complexity	⋆⋆⋆	2.66

(e) Structural Quality of Project 5

Metric Name	Rating	Rating (Interpolated)
Volume	⋆⋆⋆	3.08
Duplication	⋆⋆	2.19
Unit Size	⋆⋆	2.07
Unit Complexity	⋆⋆	2.11

(f) Structural Quality of Project 6

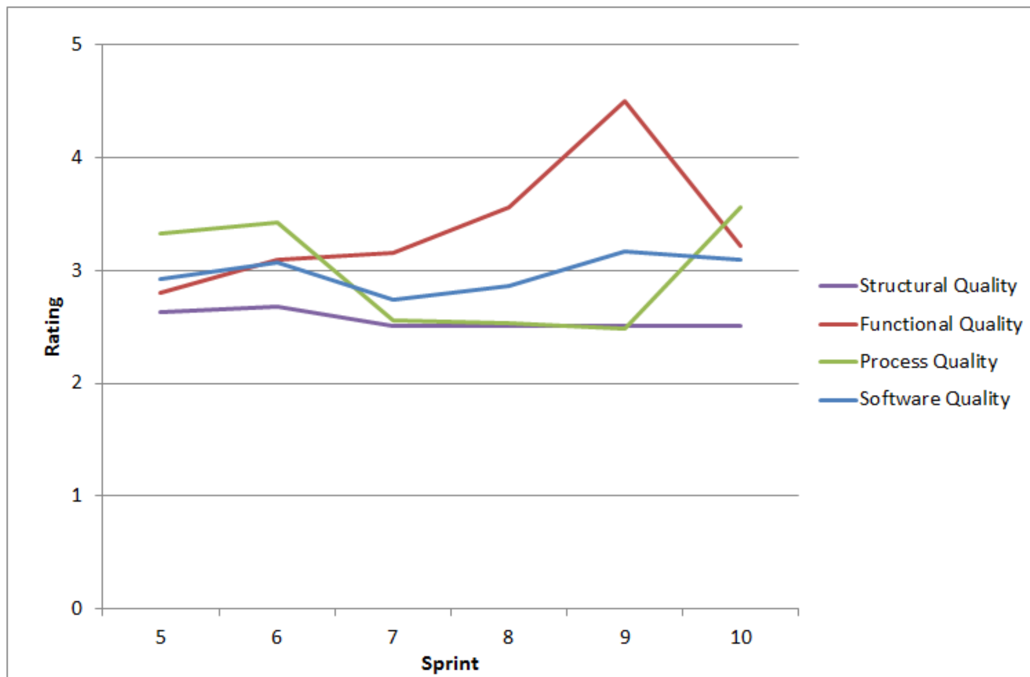


Figure 5.1: Performance Trend of Team Blue

For this purpose, they can take a look at the code metrics of the individual project. Table 5.10 shows the code metric scores for all project of the team. If we take a look at the score for project 2, which has the lowest structural quality rating, we see that it has a high amount of duplicated code. Refactoring the software by removing these code duplications can be a starting-point to increase the software's maintainability.

5.2.2 Tracking of Team Performance

Another important function of the software quality model is to track a team's performance over time. This enables a development team to analyze whether their decisions led to an performance improvement in the long run. Figure 5.1 shows a performance trend for *Team Blue*. In the diagram we recognize a slight increase in the software quality over the period of six sprints. Furthermore, we see that the change in the software quality over the six sprints is due to the variation in the functional and process quality. The structural quality only shows a small change over time.

For a more detailed analysis of the software quality and its related subcharacteristics, we refer to table 5.11 and Table 5.12 which contains the ratings for the functional and process quality of team blue.

Table 5.11: Functional Quality Trend of Team Blue

Metric Name	**Rating 7**	**Rating 8**	**Rating 9**	**Rating 10**
Enhancement Rate	2.81	3.5	4.5	3.62
Defect Removal Rate	3.5	3.62	4.5	2.81
Functional Quality	3.16	3.56	4.5	3.22

We observe the consistent improvement in the functional quality from sprint 7 to 9. The peak in the functional quality trend in sprint 9 is due to an outstanding 4.5 rating on the enhancement and defect removal rate. Although the functional quality dropped in sprint 10 again, it still represents an improvement in relation to sprint 7. The upward trend is due to the improvement in the enhancement rate rating which increased from sprint 7 to 10 by about 29%.

Table 5.12: Process Quality Trend of Team Blue

Metric Name	**Rating 7**	**Rating 8**	**Rating 9**	**Rating 10**
Velocity	1.94	3.0	3.5	3.27
Lead Time	3.08	1.49	2.75	3.62
Cycle Time	3.49	2.17	2.6	3.55
Enhancement Time Ratio	2.0	3.5	2.5	4.0
Defect Removal Effectiveness	2.25	2.5	1.05	3.37
Process Quality	2.55	2.53	2.48	3.56

But while the enhancement rate shows an improvement, the defect removal rate is declining. The team should analyze the reason for the decline. It could be the case that the quality assurance ambition is improved and consequently less bugs are reported after the delivery of a software product. In this case, no additional attention is required by the development. Whether this is really the case, you can check by comparing defect removal rate with the defect removal effectiveness from the process quality dimension (see Table 5.12). By looking at the defect removal effectiveness, we can indeed find an increase of the metric rating by about 30%.

5.2.3 Team Comparison

The software quality can also be used to compare the performance between teams. Information gained from this analysis can be used to justify the expending of extra effort on the knowledge transfer between teams, consider the composition of the teams etc. Figure 5.2 shows the software quality

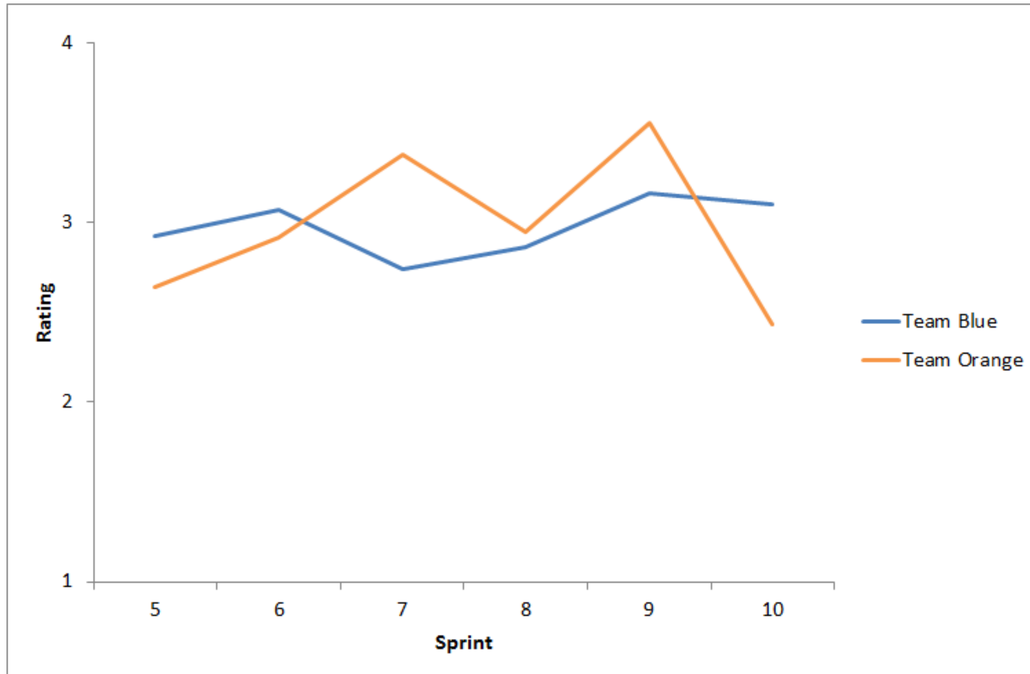


Figure 5.2: Team comparison on the basis of the software quality trend

trends over 6 sprints. The depicted diagram reveals that the software quality rating for *Team Orange* is fluctuating more heavily than for *Team Blue*. In the software quality trend of *Team Orange* we also notice a significant drop in sprint 10 which has a lower rating than sprint 5. The software quality trend of *Team Blue* shows a slight improvement over the period of 6 sprints.

For a more detailed comparison, we analyze the ratings of the software quality sub-characteristics. The functional quality of *Team Orange* in Table 5.13 also shows a high fluctuation which is the reason for the high fluctuation in the software quality. The differences between sprints are at times as high as two star ratings. The functional quality of *Team Orange* (Table 5.11) doesn't fluctuated as much.

Table 5.13: Functional Quality Trend of Team Orange

Metric Name	Rating 7	Rating 8	Rating 9	Rating 10
Enhancement Rate	1.5	3.29	2.0	2.5
Defect Removal Rate	1.55	4.35	1.97	3.5
Functional Quality	1.52	3.82	1.98	3.0

Huge performance fluctuations are not desirable because it reduces the predictability of the software development process. In some cases, this might

lead to the deferral of promised functionality which is going to lower the satisfaction of your customer. *Team Blue* better manages to keep a constant stream of new functionality.

If we compare the trend in the process quality of team orange (Table 5.14) with *Team Blue* (Table 5.12), we observe that in sprint 7 to 9 both teams have a constant process quality rating of three stars. Although, team orange scores a half star rating higher than team orange in these sprints. Furthermore, we notice that the process quality performance of *Team Blue* increased greatly in sprint 10, whereas the process quality performance dropped by one star rating. The decline of team orange is mainly due to the low velocity, lead time and cycle time ratings. This observation can be explained by a lower amount of available developer hours in sprint 10. In contrast to team blue, the sprint 10 of team blue expanded over the Christmas holidays and in this period some of the developers took a holiday break. The metric calculation doesn't account for individual holidays only for statutory holidays.

Table 5.14: Process Quality Trend of Team Orange

Metric Name	**Rating 7**	**Rating 8**	**Rating 9**	**Rating 10**
Velocity	3.02	2.5	2.38	1.39
Lead Time	3.57	3.5	5.5	1.64
Cycle Time	3.50	1.49	4.5	2.36
Enhancement Time Ratio	3.5	3.23	1.17	1.83
Defect Removal Effectiveness	2.68	4.2	1.5	2.5
Process Quality	3.25	2.99	3.009	1.95

When we look at the trend of structural quality in Table 5.15, we see that the structural quality of the teams' software products only changes slightly for the period of the 4 sprints. There are several things which explain this. First of all, the teams are fairly small and the sprint lengths are short which reduces the amount of changes in the source code during this period. Secondly, some of the projects have a high volume (number of statements) which reduces the effect of small changes in the source code on the overall structural quality of the software product.

Although, the structural quality rating for both teams are steady, we want to remark that the structural quality of *Team Orange* is rated with four stars. This is more than a star higher than the structural quality of *Team Blue*.

Table 5.15: Structural Quality Trend

Team	**Rating 7**	**Rating 8**	**Rating 9**	**Rating 10**
Blue	2.5	2.51	2.51	2.51
Orange	3.88	3.87	3.83	3.83

5.3 Model Evaluation

The software quality model is also evaluated from the user's point of view. For this purpose, we create a feedback questionnaire which is answered by employees from the software development department at AUTOonline in a semi-structured interview. The following section will discuss the design of the questionnaire. In the subsequent section we will examine the results from user feedback.

5.3.1 User Feedback Questionnaire

Riemenschneider et al. [38] have analyzed the acceptance of tools/methodologies by developers. In the study several theoretical models were analyzed concerning determinants of tool/methodology acceptance. Below you find a list of the identified acceptance determinants. The definition of the determinants were adopted from Riemenschneider et al. [38].

Usefulness "…the extent to which the person thinks using the system will enhance his or her job performance …"

Ease of use "…the extent to which the person perceives using the system will be free of effort."

Compatibility "…refers to the degree to which an innovation is perceived as being consistent with the existing values, needs, and past experiences of potential adopters."

Result Demonstrability "…refers to the degree to which an innovation is perceived to be amenable to demonstration of tangible advantages."

Perceived Behavioral Control "…refers to one's perceptions of internal or external constraints on performing the behavior …"

Subjective Norm "…the degree to which people think that others who are important to them think they should perform the behavior …"

Image "...refers to the degree to which use of an innovation is perceived to enhance one's image or status in one's social system."

Voluntariness "...defined as the extent to which potential adopters perceive the adoption decision to be nonmandatory..."

Visibility "...refers to the degree to which the results of an innovation are observable by others."

Career Consequences "...refers to outcomes that have a payoff in the future, such as increasing the flexibility to change jobs or increasing the opportunities for more meaningful work."

Some of the determinants of the original research are neglected because they are deemed not applicable for the assessment of the software quality model in this study. The reason why they are not applicable is that the determinants don't directly relate to the structural quality of the model, like *Usefulness*. For example, they relate to social factors which promote acceptance/adoption, like *Subjective Norm* or *Image*, or they relate to conditions during or after introduction, like *Voluntariness* or *Visibility*.

Questions	**Aspects**
The software quality model will enhance your job performance?	Usefulness
The software quality model provides information which supports your decision-making?	Usefulness
The metrics in the software quality model doesn't require much manual effort to measure?	Ease of use
The information which is relevant for you can easily be retrieved from the software quality model?	Ease of use
The goals of the software quality model reflect the goals of an agile software development environment?	Compatibility
The goals of the software quality model reflect the goals and principles of AUTOonline?	Compatibility
The software quality model can be used with my current knowledge?	Perceived Behavioral Control (External)
Do you think there are aspects of software quality missing in this model?	Completeness

Table 5.16: Evaluation Questions from the Feedback Questionnaire

The Table 5.16 shows the determinants and the related question which were selected for the feedback questionnaire. For each of the question with

The software quality model will enhance your job performance.

strongly agree | agree | neutral | disagree | strongly disagree

Remarks: --

Figure 5.3: Excerpt of Feedback Questionnaire

the exception of the question relating to completeness, we provide a five-point Likert scale. Figure 5.3 shows an example of a question from the feedback questionnaire. The question related to the completeness may be answered by the respondent with YES or NO. A complete version of the feedback questionnaire can be found in the appendix.

5.3.2 Results

This section we will discuss the feedback which was given by employees of the software development department at AUTOonline. A *Product Owner*, *Scrum Master* and the *Head of Software Development* took part in the feedback meeting. All participants have a background as software developer and know the scrum methodology. At the beginning of the meeting, the software quality model was presented to the participants. After the presentation the questionnaire was used to receive the feedback about the applicability of the model.

The first aspects which was covered in the interview was related to the *Usefulness* of the software quality model. The first statement, which was posed to the respondents, was "The software quality model will enhance your job performance". All three respondent agreed with this statement. The head of software development highlighted the possibility to observe the trend of the development process, team and software projects. The scrum master remarked that the model gives good possibilities to discover potential improvements which leads to faster software development.

A further statement concerning the *Usefulness* was "The software quality model provides information which supports your decision-making". The feedback on this statement was a little more diverse. The product owner rated the statement with *strongly agree* which leads to the conclusion that the model provides the information which is required by a product owner to plan a sprint. The head of software development agreed with the statement,

as well, and he further remarked that it provides the information required to decide about additional staff training or refactoring of specific software products. The scrum master responded neutral to this statement because in his opinion the scrum master isn't a decision-maker which is consistent with the traditional definition of the scrum master role.

The next aspect of the interview was the topic *Ease of use*. The first statement to which the respondents had to react was "The metrics in the software quality model doesn't require much manual effort to measure". All the respondents uniformly agreed with this statement. Although the product owner added that in general time logging is considered as waste by developers in an agile environment. The next statement for the same aspect was "The information which is relevant for you can easily be retrieved from the software quality model". The respondents uniformly agreed with this statement. The product owner mentioned that it is able to reveal interesting information but you might need additional information from outside the model to explain certain measurement results. For example, it is important to know the point in time in which team composition changed, because this has an influence on the team's velocity.

The first statement for the *Compatibility* aspect was "The goals of the software quality model reflect the goals of an agile software development environment". The opinion of the respondents about this statement differs. The head of software development strongly agrees with this statement. The scrum master agrees with this statement because he sees that it supports continuous improvement which is an agile development goal. Whereas the product owner responds with *neutral* to this statement. He explained that the measurement goals were deduced from the company's goals but this may not coincide with agile development goals in general. With the other statement for this aspect "The goals of the software quality model reflect the goals and principles of AUTOonline" all respondents uniformly agreed again.

The next statement posed to the interviewees was "The software quality model can be used with my current knowledge". The head of software development and scrum master agree with this statement and the product owner even rates the statement with *strongly agree*.

The last statement "Do you think there are aspects of software quality missing in this model" relates to the *Completeness* of the software quality model. The scrum master mentioned that you could add metrics relating to test coverage in the structural quality part of the model. The head of software development proposed to measure the cycle time metric differently. He said that it is better to measure the cycle time from the *In Work* date instead of the *Creation Date* because sometimes user stories aren't picked up for a while because they have a low priority. Furthermore, did he propose to

measure how many percent of the known bugs were fixed in a sprint.

From the feedback meeting we conclude that the model was deemed to be useful to increase the job performance of the respondents and it delivers information required for the daily operations of the respondents. Only the scrum master rated lower than the other two respondents. But this is due to perception that the scrum master isn't a decision-maker. This is true for the sprint planing in which it is decided what user requirements are going to be implemented. But he also has the responsibility to identify impediments which obstruct team performance. Information about impediments can be taken from the model, like low software maintainability. We think that the scrum master will realize this potential once he uses it for a longer period of time.

Furthermore, can we conclude that the respondents judge the software quality model as easy to use. The data for the software quality metrics are already available at AUTOonline and therefore the additional effort required for the measurements are minimal. In addition, the participants perceive that they can easily retrieve and interpret the data which they need from the model. The only criticism relates to the logging of expended developer hours. Although this is not a problem at AUTOonline, because the expanded developer hours are already being logged, you might need to spend additional effort for logging in other companies where this information is not available in the ITS.

The measurement goals of the software quality model are perceived to be compatible with the AUTOonline goals and principles. Two of the respondents also see the compatibility with the agile principles. Only the product owner deviates from this judgment. In practice, you will always find small deviation from the Scrum methodology which explains the differences between the general agile software development goals and the concrete goals of a company. After all, Scrum is a methodology with a set of tools and guidelines which are supposed to improve the software development performance of a company. The components of the Scrum methodology are taken by companies and adjusted to the specific needs of company.

Concerning the *perceived behavioral control*, can we conclude that all respondents feel that they can use the model with their current knowledge without much extra learning effort. Finally, we can summarize that three changes and extensions of the model were proposed. It was proposed to extend the structural quality measurement with metrics to rate the test quality, to add a metric to analyze how many percent of the known bugs were fixed in a sprint and to change the measurement of the cycle time.

Chapter 6

Conclusion and Future Work

The main objective of this thesis was to find an answer to the question *how software measurements can be incorporated in an agile software development process to support continuous improvement*. For this purpose we investigated an approach that can be followed to define a measurement model. The identified approach was used to define a model to assess the software quality of an agile software development process. Furthermore, we applied the software quality model in the business context of the AUTOonline organization in order to evaluate the usefulness of the model. In this chapter, we summarize the findings of this study and we indicate possible areas for future research.

6.1 Conclusions

This section summarizes the findings of this study and provides answers to the research questions.

RQ1: What approach should be followed to define a software measurement model in an agile environment?

The first goal was to identify a structured approach which can be used to establish a software measurement program. The GQM approach proved to be suitable to design a software measurement program in this study. The pivotal factor is that the GQM approach follows a top-down fashion for the definition of a measurement model. The construction of a measurement model starts with the identification of the business goals. The business goals are then linked to measurement goals. From the measurement goals we derive questions which need to be answered to assess the development performance against the measurement goals. The answers to these questions are provided by software metrics. The consideration of the business goals in the choice of

the metrics makes the GQM approach suitable because it guarantees that the model measures something that is relevant for the success of the company.

In this study the GQM approach was combined with a rating system that facilitates the interpretation of metric scores and helps to identify parts of a system which may cause a problem. The method to establish the rating system in this study is based on previous research of SIG. The ratings in this method are assigned according to statistic analysis of a metric's data distribution.

RQ2: What prerequisites must be fulfilled to ensure that the software measurement model delivers significant information?

During the selection of metrics we have to pay attention to certain factors which influence the success of a measurement model. Some of these factors relate to the selection and combination of the metrics in a measurement model. These factors can be described in the form of pitfalls by the names of *Metric in a Bubble*, *Treating the Metric*, *One-Track Metric* and *Metrics Galore*.

Metric in a Bubble describes a situation in which the stakeholders of a measurement model aren't able to determine what a metric score means. *Treating the Metric* occurs when the people that are being measured alter their behavior just to score well on a metric without really satisfying the business goal. Every business goal consists of several dimensions. The measurement model won't support an improvement with respect to the business goals if you fail to adequately cover all dimensions(*One-Track Metric*). But at the same time the measurement model must prevent the *Metrics Galore*. In this situation stakeholders of the measurement model get overwhelmed with a huge amount of metrics. This bears the risks that the measurement model is ignored entirely.

Further prerequisites are that the data collection for the selected metrics can be automated as far as possible and that the presentation of the metrics enables a root-cause analysis.

RQ3: What are the measurement goals in an agile software development process?

The goal of software development is to ensure high quality software. Software quality is associated on the one hand with the quality of the software products and with the quality of the development process on the other hand.

The quality of the software products can be further broken down into their functional and structural quality. The functional or external quality

is the quality which is perceived on the outside of a software product. The functional completeness and correctness constitute the functional quality and determine the value which a user receives from a software product. The structural or internal quality is related to the source code of the software products. The higher the structural quality of a software product, the easier the software product can be maintained.

Process quality is connected to the goal of speeding up the development process and establishing a viable platform for future software enhancements [25]. The higher the process quality, the higher is the ability to react to changes in technology or user requirements in a timely manner.

RQ4: How can the metrics be effectively incorporated in an agile software development process?

Using a measurement model, software quality can be reviewed at the end of every iteration. In the review the development team should analyze the performance of the past iteration, identify potential impediments to a better performance and plan actions to remove these impediments in order to improve future performance. Moreover, the software quality model can be used in regular intervals to assess the performance trend of a team to see whether the taken actions led to a performance improvement in the long perspective or not. Managers can also use this model to compare the performance of the development teams. Information gained from the team comparison can, for example, be used to change the composition of the development teams, decide on investments in staff training or knowledge transfer.

RQ5: How useful is the proposed software measurement model in an agile environment?

We have conducted a user feedback interview, after the application of the software quality model to the business context of this study. The result of the feedback suggests that the software quality model and the related metrics are applicable to the business context. Some extension and adjustment in the metrics of the model were mentioned, but the respondents deemed that the model enables a fair assessment of software quality. The user feedback interviews have to be repeated with a higher amount of respondents in order to confirm the results.

From the feedback we also conclude that the prerequisites for a successful introduction of the software quality model are fulfilled and therefore the software quality model is applicable to other software development environments. Some adjustment on the metric level may be necessary to accommodate the

special circumstances in a different development environment. The flexibility of the model enables us to make the required changes.

Moreover, do we need to recalibrate the rating system of the measurement model with more data to receive more viable metric thresholds.

6.2 Contributions

In this section we discuss the contributions of this study. We defined three contributions: A framework for the definition of a software measurement program, definition of a model to measure the software quality of an agile development process and performing a case study to analyze the applicability of the model in practice. These contributions are explained in more detail in the following list:

A framework for the definition of a software measurement program. We have shown how a measurement model can be defined in a top-down fashion based on the GQM approach. For the choice of the metrics we have provided a guideline concerning the combination of metrics which help to ensure the successful implementation and introduction of a measurement program.

Definition of a model to measure the software quality of a agile development process.

In this study we have used the framework for the definition of a software measurement program to establish a model to measure the software quality in an agile software development environment. The GQM approach which was used to define the software quality model together with the rating system facilitates the interpretation of the metrics scores. The facilitated interpretability of the metric scores prevents the undesirable consequences of the *Metric in a Buble* and *Treating the Metric*. The software quality model contains the break down of software quality into sub-characteristics which can be measured using the proposed metrics and the corresponding rating system. During the selection of the metrics, we paid attention not to cover only one aspect of software quality (*One-Track Metric*) and at the same not to include too many metrics (*Metrics Galore*). The flexibility to add or remove characteristics and metrics to the model ensures that we can adjust the model to the specific needs of a software development environment. All the afore mentioned factors increase the probability of a successful application of the software quality model.

Performing a case study to analyze the applicability of the model in practice. We conducted a case study by applying the software quality model to the business context of this study. The case study showed how useful information can be retrieved from the model. The information from the model was used to assess the team performance for a particular sprint, analyze the performance trend of a development team and compare the performance of different development teams. Moreover, we elaborated how this information can be utilized to identify impairments of the product and process quality. The identified impairments serve as a starting point for a team to improve their future performance.

6.3 Threats to Validity

During the course of this study we have identified factors which can potentially weaken the validity of the information that is retrieved from the software quality model and limit the generalizability of the software quality to other environments. The discussion of the validity threats will be separated into *Construct Validity*, *Internal Validity* and *External Validity* as proposed by Perry et al [36].

6.3.1 Construct Validity

Does the proposed measurement model accurately model software quality in a software development environment?

Software Quality Measurement

The software quality model was established by using the GQM approach. The aspects which are attributed to software quality were identified from the literature. The software quality model concentrates on the most important aspects which were identified. During the selection we paid attention to the prerequisites for software metrics described in Section 3.1. This way we guarantee a fair assessment of software quality and at the same prevent that the users of the model get overwhelmed by the amount of available metrics. Nevertheless, the model is not complete. Additional measurements can be integrated into the model, once the software quality is successfully integrated into the development process and the development team has become acquainted with using the software quality to improve their performance. A potential extension of the structural quality aspects with metrics for test code

quality was already mentioned during the user feedback. The flexibility of the model can be seen as strength of the model because no one environment completely equals another. Through the ability to add and remove metrics we can adjust the model to the specific needs of an environment.

Quality of Data

Some of the metrics from the software quality model were calculated using data from ITS. The accuracy of the data in the ITS is dependent on the usage of the ITS. Some metrics, like the cycle time, relate to the issue life cycle. In some cases issues (user stories) skipped over some stages in the life cycle. For example, they didn't enter the *Ready for Deployment* state but were directly transferred to the *Closed* state. This could be due to a mistake of the ITS user but can also occur intentionally. An implementation of a user story can happen alongside with the deployment. For example, the change of a configuration in a software doesn't require an additional deployment step. Implementing changes in the configuration and saving them is equivalent with their deployment. This is the reason why we took the date when the issue entered the *Closed* date for the calculation of the cycle time.

Furthermore, it is important to note that we only analyzed the source code which was written in C#. For example, JavaScript was not considered in the source code metrics. Thus, software that which has a significant amount of program logic implemented in JavaScript, scores better in the structural quality than if that logic was implemented in C#. A detailed analysis showed that most JavaScript code consists of standard libraries which were not developed by AUTOonline. Most projects use a negligible amount of JavaScript code that was implemented at AUTOonline. The only two projects that use a significant amount of internally developed JavaScript is Project 4 and Project 7. Project 4 contains 10916 lines of JavaScript code and Project 7 contains 5710 lines of JavaScript code. This circumstance has to be considered when comparing the maintainability of the software products.

6.3.2 Internal Validity

Can changes in the dependent variables be safely attributed to changes in the independent variables?

Confounding Factors

In the metric calculation of software quality we account for certain factors, like the sprint length. The reason for this is that not sprints have the same

amount of working days. The length of a sprint clearly has an effect on the amount of user story points which a development team can accomplish during a sprint. By normalizing the metrics with the sprint length, we achieve a better comparability between sprints. But in order to draw the right conclusion from the measurement data we need to consider other factors. An important factor to regard is change in the development staff. Sometimes a development team accomplishes less user story points during a sprint, if a member of the team is replaced by new developer. The number of developers on the team stayed the same but the team needed to spent time for the training of the new team member which leaves less time for the implementation of user stories. Therefore, the development team isn't directly responsible for the decrease in the performance trend.

When the software quality model is used to compare the performance of different development teams, we need to consider another factor. This factor is the experience of a development team. For example, a highly experienced team is able to accomplish more user story points than a lesser experienced team.

Yet another factor which influences the comparability of development teams is the difference in user story point estimations. A team which tends to overestimate the complexity of user story, will implement more user story points than a team which underestimates the complexity.

Low Amount of Data Points in Benchmark

For the calibration of the metric thresholds we used the data for 6 sprints of two development teams which were taken from AUTOonline. This gives us 12 data points in the benchmark for the metrics which are directly calculated for the sprint level, like the velocity metric. The low amount of data points has the consequence that outliers have greater effect on the selection of the metric thresholds. Consequently, the thresholds may change considerably when the model is recalibrated with additional data points.

6.3.3 External Validity

Can the study results be generalized to settings outside the study?

Generalization to other Agile Methodologies

The software quality model uses terminology of the Scrum methodology in the definition of the metrics. Although, these terms may be unique to Scrum environments, we can find the concept behind them in all agile environments.

The term *sprint* is a special Scrum term to describe the iteration concept in a software development process. Some of the agile methodologies don't explicitly include the iteration concept, like Kanban, but they still don't neglect the concept. The *7th Annual State of Agile Development Survey* conducted by *VersionOne* with 4,048 respondents showed that 75% use iteration planning [41].

The term *user story* is also used in other agile methodologies, like Extreme Programming, but we could also replace it with the term *user requirement*. The calculation of the development capacity (velocity) in the software quality is based on *story points*. Story points are unique to Scrum where they are used to assign complexity estimations. Other agile environments use ideal programming days or hours as a unit for estimations. If the software quality model is applied to these environments, the cycle time metric can use the ideal programming days as normalizing factor instead of story points.

Hence, when the software quality model is applied to other agile environment the general structure of the model can be adopted but some of the metric's calculation need to be adjusted.

Generalization to other Development Environments

The functional and process quality metrics were derived from the research based on the ITS which is used in the business context. The existence of an ITS is therefore a prerequisite for using the functional and process metrics from the software quality model. Furthermore, the enhancement time ratio uses the logged hours of an ITS issue as a normalizing factor. This was done to analyze the actual effort that was expended on user stories, bugs and tasks. In an environment in which this information is not available or costly to retrieve we can replace it with the enhancement ratio metric.

6.4 Future Work

In this section we will discuss how the research of this study can be continued.

Model Extensions

We already indicated that the software quality model enables a fair assessment of the software quality by considering the most important aspects of software quality. But the model can be extended with additional metrics to accommodate additional aspects of software quality.

The structural quality assessment concentrates on a subset of the maintainability characteristics which are included in the SIG Maintainability

Model and ISO 25010. Additional metrics can be integrated to analyze additional characteristics, like *Modularity* and *Reusability*, as well as to gain a more profound insight into the characteristics of the software maintainability that are already present in the model. A starting point for the extension was already identified in the user feedback interview. One respondent mentioned that the assessment of test coverage was missing in the model. This aspect relates to the *Testability* characteristic of software maintainability. For this purpose, our research can be joined with the research of Athanasiou about the construction of a test code quality model [7].

Further metrics could be integrated to complement functional and process quality. For example, the functional quality characteristic could be extended with a metric to measure the amount of bugs left in the product backlogs at the end of a sprint. The process quality model could then be extended with a metrics which calculates how many percent of the bugs from a backlog were fixed in a sprint. These last two examples were proposals coming from the respondents in the user feedback.

Data Set Extension and Correlation Testing

The calibration of some metric thresholds was based on a small amount of data points in the benchmark. Data points of additional sprints should be collected and integrated into the benchmark. Consequently, the metric thresholds should be recalibrated. The reliability of metric thresholds can be increased this way. In a next step correlation tests can be executed. In the correlation tests it would be interesting to analyze correlations between the different software quality sub-characteristics. For example, the correlation of structural quality with process quality or the correlation of process quality and functional quality.

Repeating User Feedback

The feedback interviews should be repeated with additional respondents to confirm the results from the user feedback presented in this study. It would also be interesting to fully integrate the model into the development process and then reevaluate the applicability of the software quality model. The current evaluation was based on the estimations of the respondents concerning the ease of use. If the respondents have actually used the model in the daily operations, they could give a more profound assessment.

Linking Measurement Goals to Throughput Accounting

In the beginning of the study we discussed the linkage between the business goals and the software measurement goals. The software quality model concentrates on the intangibles which the software quality adds to the business value. Further research can extend the link to the tangibles by incorporating the monetary value (throughput) which the implementation of a particular user requirement adds to the business value.

Bibliography

[1] ACM. Computing degrees & careers. http://computingcareers.acm.org/?page_id=12, 2006. Retrieved: 16 March 2013.

[2] Agile Alliance. Guide to agile practices. http://guide.agilealliance.org/.

[3] Scrum Alliance. Scrum's three roles. http://www.scrumalliance.org/pages/scrum_roles. Retrieved: 11 October 2012.

[4] T L Alves, C Ypma, and J Visser. Deriving metric thresholds from benchmark data, 2010.

[5] D. J. Anderson. *Agile management for software engineering: applying the theory of constraints for business results*. Prentice Hall, 2004.

[6] Tony Aquila. Communicate your vision. http://www.nyse.com/pdfs/ForStakeholders.pdf, 2008. Retrieved: April 2013.

[7] Dimitrios Athanasiou. Constructing a test code quality model and empirically assessing its relation to issue handling performance. Master's thesis, Delft University of Technology, Netherlands, 2011.

[8] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[9] V.R. Basili, M. Lindvall, M. Regardie, C. Seaman, J. Heidrich, J. Munch, D. Rombach, and A. Trendowicz. Linking software development and business strategy through measurement. *Computer*, 43(4):57 –65, April 2010.

[10] Kent Beck et al. Agile manifesto. http://agilemanifesto.org/principles.html, 2001. Retrieved: 9 October 2012.

[11] Kent Beck et al. Principles behind the agile manifesto. http://agilemanifesto.org/, 2001. Retrieved: 9 October 2012.

[12] Dennis Bijlsma. Indicators of issue handling efficiency and their relation to software maintainability. Master's thesis, University of Amsterdam, Netherlands, 2010.

[13] Dennis Bijlsma, MiguelAlexandre Ferreira, Bart Luijten, and Joost Visser. Faster issue resolution with higher technical quality of software. *Software Quality Journal*, 20(2):265–285, 2012.

[14] Eric Bouwers, Joost Visser, and Arie van Deursen. Getting what you measure. *ACMQUEUE*, 2012.

[15] David Chappell. The business value of software quality. http://www.davidchappell.com/writing/white_papers.php.

[16] David Chappell. The three aspects of software quality: Functional, structural, and process. http://www.davidchappell.com/writing/white_papers.php.

[17] W. Bruce Chew. Basic operations self-instructional workbook. http://hbswk.hbs.edu/archive/1460.html, April 2000. Retrieved: 10 October 2012.

[18] Thomas Corbett. Making better decisions. *CMA Management*, 73(9):33, 1999.

[19] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 357–370, New York, NY, USA, 2000. ACM.

[20] Martin Fowler. The new methodology. http://martinfowler.com/articles/newMethodology.html. Retrieved: 10 September 2012.

[21] Martin Fowler. Continuous integration. http://www.martinfowler.com/articles/continuousIntegration.html, May 2006. Retrieved: April 2013.

[22] E.M. Goldratt. *The Haystack Syndrome: Sifting Information Out of the Data Ocean*. North River Press, 2006.

[23] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A Practical Model for Measuring Maintainability. In *International Conference on the Quality of Information and Communications Technology*, pages 30–39, 2007.

[24] Jim Highsmith. What is agile software development? *CrossTalk*, 15(10):32, October 2002.

[25] Jim Highsmith. Beyond scope, schedule, and cost: The agile triangle. http://jimhighsmith.com/beyond-scope-schedule-and-cost-the-agile-triangle/, November 2010. Retrieved: 24 October 2012.

[26] Jim Highsmith. Velocity is killing agility! http://jimhighsmith.com/velocity-is-killing-agility/, November 2011. Retrieved: 10 October 2012.

[27] Jim Highsmith. Determining business value. http://jimhighsmith.com/determining-business-value/, January 2013. Retrieved: 24 February 2013.

[28] ISO. Iso/iec 25010: Systems and software engineering systems and software quality requirements and evaluation (square) system and software quality models, March 2011.

[29] C. Jones. Software defect-removal efficiency. *Computer*, 29(4):94–95, Apr.

[30] Corey Ladas. Lead time vs cycle time. http://leanandkanban.wordpress.com/2009/04/18/lead-time-vs-cycle-time/, 2009. Retrieved: 3 April 2013.

[31] Klaus Leopold. Das etwas andere kochrezept. http://www.heise.de/developer/artikel/Software-Kanban-im-Einsatz-1235465.html, 2011. Retrieved: December 2012.

[32] Eliyahu M. Goldratt. *What is this thing called Theory of Constraints an how should it be implemented?* The North River Press, 1990.

[33] William F. Nazzaro and Charles Suscheck. New to user stories? http://www.scrumalliance.org/articles/169-new-to-user-stories, April 2010. Retrieved: 11 October 2012.

[34] B.E. Needles, M. Powers, and S.V. Crosson. *Principles of accounting*. Houghton Mifflin, 2008.

[35] Frank Niessink and Hans van Vliet. Measurement program success factors revisited. *Information and Software Technology*, 43(10):617 – 628, 2001.

[36] Dewayne E. Perry, Adam A. Porter, and Lawrence G. Votta. Empirical studies of software engineering: a roadmap. In *In Proc. of the conference on The future of Software engineering*, 2000.

[37] Roger S. Pressman. *Software Engineering: A Practioner's Approach*. McGraw-Hill, 2005.

[38] C.K. Riemenschneider, B.C. Hardgrave, and F.D. Davis. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *Software Engineering, IEEE Transactions on*, 28(12):1135–1145, 2002.

[39] Goldratt UK. Throughput accounting. `http://www.goldratt.co.uk/resources/throughput_accounting/index.html`. Retrieved: April 2013.

[40] R. Van Solingen and E. Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, 1999.

[41] VersionOne. 7th annual state of agile development survey. `http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf`, 2013. Retrieved: May 2013.

[42] Joost Visser. How does your software measure up?, 2012.

[43] Bill Wake. Invest in good stories, and smart tasks. `http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/`, August 2003. Retrieved: 11 October 2012.

Glossary

Backlog grooming: Backlog grooming is an activity to clean up the product backlog. In the process of backlog grooming new issues are created according to newly identified requirements. Moreover, issues from the issue tracking system are revisited. Issues that have become obsolete are removed. Effort estimates and priorities are added or updated if necessary.

Continuous Integration "Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible" [21].

Customer Value: Customer value is the value received by the end-customer of a product or service.

Product Owner: The product owner (PO) is responsible for the business value of a project. The PO decides in cooperation with the customer what will be built.

Refactoring: Refactoring is a technique for improving the structure of existing source code without altering the external behavior.

Scrum Master: "The Scrum Master is a facilitative team leader who ensures that the team adheres to its chosen process and removes blocking issues" [3].

Sprint: A sprint is an iteration of work in agile software development during which an increment of a software product is implemented.

Story point: Is an unit to represent the complexity of a user story. In the planning meeting, the development team assigns story points to a user story to show how complex it is to implement this particular user story.

User story: "User stories are actually narrative texts that describe an interaction of the user and the system, focusing on the value a user gains from the system" [33]. Wake has created the acronym *INVEST* to describe the characteristics of a good user story [43]. Wake's definitions for the INVEST attributes are as follows:

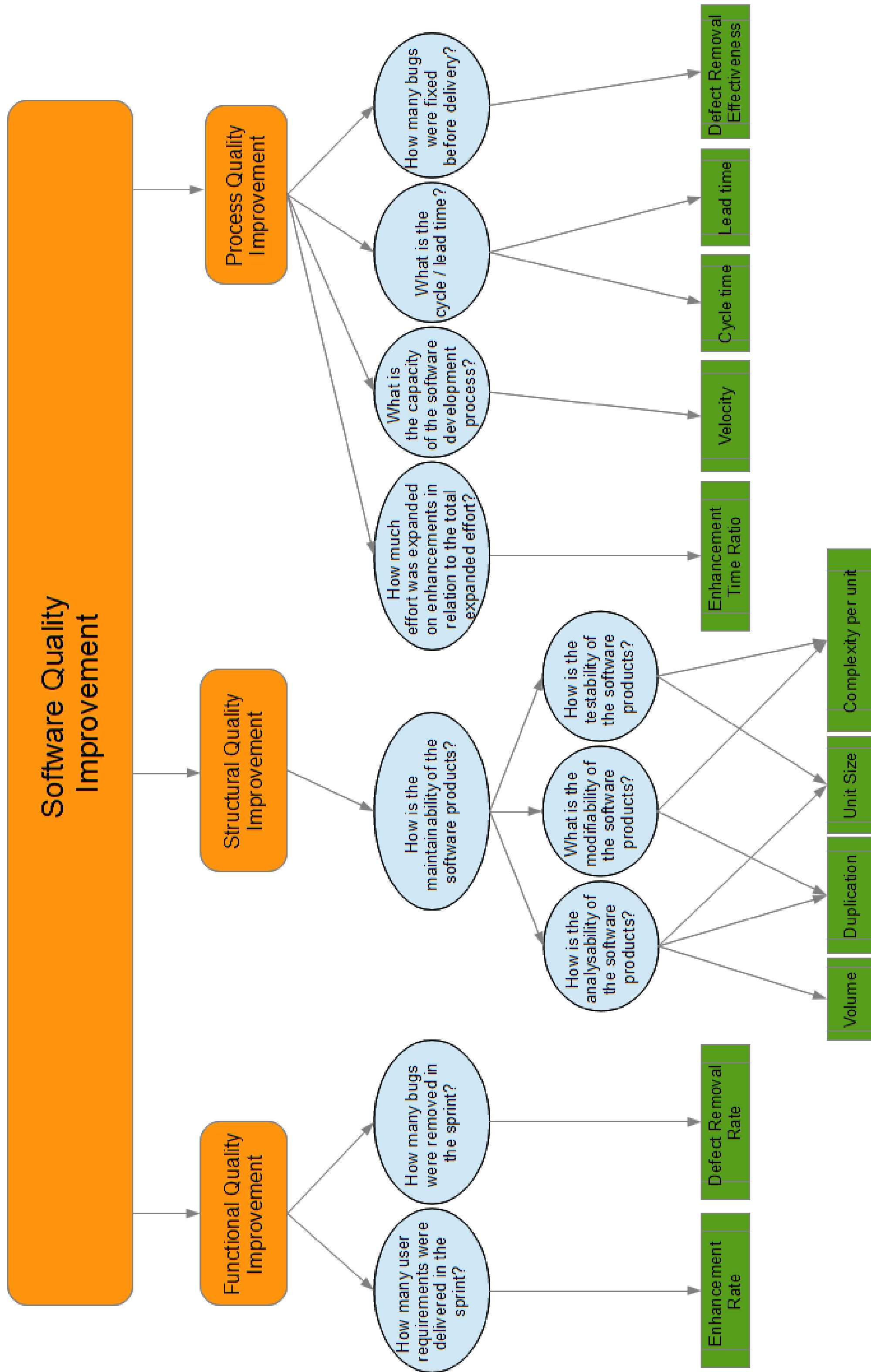
- ***I*ndependent** Stories should be independent from each other, so that they can be scheduled and implemented in any order.
- ***N*egotiable** A story captures the essence, not the details. The details will be worked out by the development team in cooperation with the customer.
- ***V*aluable** A story needs to provide value to the customer.
- ***E*stimable** A story needs to contain the information required to prioritize and schedule the story.
- ***S*mall** A story should be small as possible to make the story's scope palpable and to make estimations easier.
- ***T*estable** Acceptance criteria help understand the goal of the user story.

Variable Cost: "A variable cost is a cost that changes in direct proportion to a change in productive output (or some other measure of volume" [34].

Velocity: "At the end of each iteration, the team adds up effort estimates associated with user stories that were completed during that iteration. This total is called velocity" [2].

Appendix A

Software Quality Model (GQM)


Software Quality Improvement
Process Quality Improvement
Structural Quality Improvement
Functional Quality Improvement
How many bugs were fixed before delivery?
What is the cycle / lead time?
What is the capacity of the software development process?
How much effort was expanded on enhancements in relation to the total expanded effort?
How is the maintainability of the software products?
How is the testability of the software products?
What is the modifiability of the software products?
How is the analysability of the software products?
How many bugs were removed in the sprint?
How many user requirements were delivered in the sprint?
Defect Removal Effectiveness
Lead time
Cycle time
Velocity
Enhancement Time Ratio
Complexity per unit
Unit Size
Duplication
Volume
Defect Removal Rate
Enhancement Rate

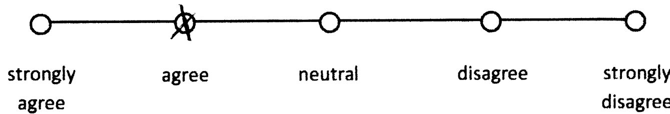
Appendix B

Feedback Questionnaire

Name:

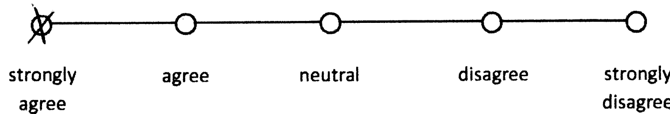
Job Title / Scrum Role: Product Owner

The software quality model will enhance your job performance.



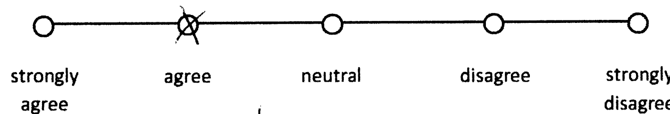
Remarks:

The software quality model provides information which supports your decision-making.



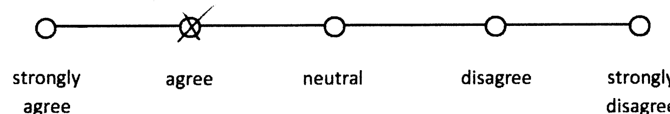
Remarks: I bet this will reveal an optimum velocity over a short time on a team based view

The metrics in the software quality model doesn't require much manual effort to measure.



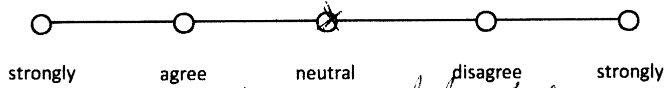
logging time actually is considered work for a developer

The information which is relevant for you can easily be retrieved from the software quality model.



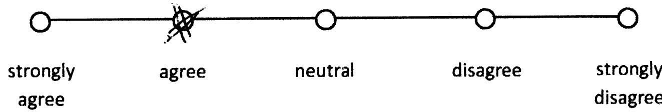
including some more external information it can reveal interesting correlations

The goals of the software quality model reflect the goals of an agile software development environment.

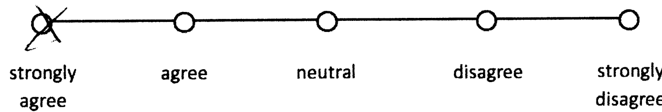


the goals were deduced from the company's goals this may not always result in agile goals

The goals of the software quality model reflect the goals and principles of AUTOonline.



The software quality model can be used with my current knowledge.



Do you think there are aspects of software quality missing in this model.

☒ No ☐ Yes: until proven wrong

there are for sure some missing but this needs more empirical values.