# Improving Code Quality in Agile Software Development

*Version of October 20, 2018*

Lars Krombeen

# Improving Code Quality in Agile Software Development

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Lars Krombeen
born in Delft, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Capgemini
Reykjavikplein 1, 3543KA
Utrecht, the Netherlands
https://www.capgemini.com/

*“Every block of stone has a statue inside it and it is the task of the sculptor to discover it.“*

- Michelangelo

# Improving Code Quality in Agile Software Development

Author : Lars Krombeen
Student id : 4280709
Email : l.krombeen@student.tudelft.nl

**Abstract**

Agile software development is a popular approach for developing software. Another important topic of research in software engineering is code quality. Unfortunately, a minimal amount of extensive research has been done on how these two influence each other. The goal of this study is therefore to explore the connection between these two using a qualitative approach. To understand this connection we will use Grounded Theory as a qualitative methodology to interview 20 participants across two organisations. In doing so we present a detailed description of Grounded Theory implementation and the results we obtain from it. The results are used to explore the relation between code quality and agile software development. The results show that team empowerment is the core relation between them. The results are structured in a theory which establishes four core values for achieving team empowerment, conditions that apply to these values and which practices can be applied to stimulate the conditions. The outcomes of the study are further verified using an online questionnaire across multiple countries. The theory will be expanded further to establish theoretical links between Agile best practices and code quality metrics to give teams concrete solutions to improve their code quality scores.

**Keywords:** Empirical research, Software engineering, Grounded Theory, Code quality, Agile software development, Team empowerment

Thesis Committee:

| | |
|---|---|
| Chair | : Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft |
| University supervisor | : Dr. Ir. F.F.J. Hermans, Faculty EEMCS, TU Delft |
| Company supervisor | : D. Fraser, Capgemini |
| Committee Member | : Dr. W.P. Brinkman, Faculty EEMCS, TU Delft |

# Preface

Before you lies the resulting report of a project that lasted from March 2018 to October 2018 for obtaining the degree of Master of Science in Computer Science. Even though this work might not be that typical for a Computer Science student, it made me learn a lot about conducting rigorous and structured research. Moreover, I was able to improve social skills such as taking interviews, which is less focused on at the university's department of Computer Science. There are a couple of people I would like to thank for making this research possible.

First of all, I want to thank Felienne Hermans for her constant supervision and 24/7 availability. Finding a topic for my thesis was difficult and being able to brainstorm with you greatly helped. Furthermore, I would like to thank you for your counselling and supervision during the project. It aided my capabilities as a researcher and provided insights I would have possibly overlooked. Secondly, I want to thank Arie van Deursen because, in the meetings we had, you gave me insights into directions the research could go. Next, I would like to thank Sohon Roy. As an inexperienced researcher who never conducted interviews you helped me in getting started. The tips about interviewing and the feedback you gave on my initial questions helped in improving the quality of the interviews.

None of this project would have been possible if Frank Singendonk would not have reached out to me, so I would like to thank him for this great opportunity. Furthermore, I want to thank Desiree for giving me supervision during the internship. With the full freedom you offered me and the great opportunities you gave me, this research could be executed smoothly.

I want to thank everyone that participated in the interviews. Without you, the research would not have been possible. In addition, I would like to thank the respondents of the questionnaires and those who helped in refining the draft. Without your responses, the research would lack some serious grounds.

Last but not least, I want to thank my family and friends for their support. Being able to explain my problems and difficulties I faced during my thesis solved most of them. Thank you for listening to my endless dialogues about my study. But know that you have done so in the name of science.

Enjoy reading!

Lars Krombeen
Delft, the Netherlands
October 20, 2018

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

When developing software, customers usually want products of high quality. Characteristics such as security and maintainability play a main factor in determining if the developers have produced high-quality software [36]. Different tools aid developers in producing this by calculating a statistical analysis of the code and indicate critical errors and how to solve them.

Code quality has been a topic that has been researched frequently. Studies report on factors that cause software to be of high or low quality and the way in which this affects the business value, or what risks these factors can cause [9]. Other studies take a more technical view and discuss what good quality means [11, 52]. Nagappan et al. [44] researched the effects of the organisational structure on code quality.

In a world where Agile software development and Scrum become more and more popular, processes are being changed which could affect code quality. Unfortunately, studies on the impact of code quality in relation to Agile software development are scarce. The work of Nagappan et al. [44] is one of the few studies that is related to this thesis, but it has not included the software methodology factor such as Agile software development in the organisational structure. Multiple Agile software development frameworks/methodologies exist that allow developers to work in an Agile way. In these, the creators state how their methodology can cause the product quality to improve. An example is the *Definition of Done* which is a property of the Scrum framework [55]. The Definition of Done ensures that the product has the quality the team agreed upon.

In this paper, we aim to understand the relation between code quality and agile software development by using a qualitative research on the relation between them. These relations are studied using *Grounded Theory*, which is a research methodology. Using this we can explore and define new theories about the relations in a qualitative phase using interviews. Although Grounded Theory was originally designed for researching social sciences, successful implementations have been made in the field of computer science [3, 34] and discussed by Stol et al. [59]. The Grounded Theory is further verified using a quantitative phase which makes use of a questionnaire. We will take this one step further and design a practical solution to improve code quality in Agile software development that provides insights for developers on how they can adjust their processes to improve their code quality metrics.

The contributions of this thesis are as follows:

- A theoretical model that explains the relations between code quality and Agile software development.
- A quantitative survey that further verifies the qualitative results from the Grounded Theory.
- A practical solution for developers to build code with a higher quality in Agile software development or to improve their code quality metrics.

## 1.1 Research Questions

The first step in developing a practical means to increase the code quality of Agile software developers requires us to research how code quality and Agile software development are related. Since there is limited relevant literature on this topic we perform a qualitative exploratory study to establish this relation followed by a quantitative phase to verify the general applicability of the theory.

> **RQ1:** What is the main phenomenon and its relations that explain the relation between code quality and Agile software development and how does this explain it?

The results of the theory are only applicable to the sample set of the interviews. Moreover, although the theory will be correct, it will most likely not be complete. The next step is to use the results and compare it to a larger sample set. This will increase the applicability of the results. However, it will not expand the theory. Thus, our second research question is focused around verification of RQ1 and is as follows:

> **RQ2:** How does the theory about code quality in Agile software development compare to global experience?

The results from the first two research questions are based on interviews and interpretations. Based on the results a theory will be defined, indicating what is important to do in Agile software development which provides basic insights developers can use to increase their code quality. The theory consists of different layers where the bottom layer contains practices. These practices are analysed in which way they impact code quality and which implementations enhance code quality. Moreover, the analysed data from the interviews are complemented with practices we find in literature that fit into the model. Our third research question is therefore:

> **RQ3:** Which Agile software development practices impact code quality and how should they be implemented to enhance code quality?

Having established best practices that impact code quality containing a description of how they are best implemented to enhance code quality has not given us a practical solution to the problem. Therefore, we take a pragmatic approach to bring the results back into practice. We will do this by linking the best practices to subcharacteristics of the ISO/IEC 25010:2011 standard. This gives developers an idea of what processes they can change to improve their scores.

> **RQ4:** What are best practices in Agile software development methodologies, how are they linked to (sub)characteristics of code quality and how can they be used to increase quality scores through enhancing Agile processes?

## 1.2 Chapter Overview

The remainder of this paper is structured as follows. Chapter 2 contains a short literature survey that describes basic aspects about Agile software development, code quality and the relation between them. We will describe Grounded Theory in Chapter 3. Chapter 4 will further explain the goal of our research and provides an overview of our experimental design to answer our research questions. The results of the Grounded Theory and the resulting theory are established in Chapter 5 and provide a theory that answer RQ1. Chapter 6 continues on the Grounded Theory and conducts our quantitative phase to further verify the theory and answer RQ2. The results are given a pragmatic approach to bring the theory and reality together in Chapter 7 to develop a practical use of the model. Best practices that answer RQ3 are given and linked to metrics from the literature survey to answer RQ4. We revisit our research questions in Chapter 8 which will be followed by a discussion in Chapter 9.

# Chapter 2

# Relation Between Code Quality and Agile Software Development

We start this thesis with a short literature survey. The goal of this survey is to get a basic understanding of the rules of Scrum and to give a definition of Code Quality. Once the concepts of our study have been defined we reviewed literature that discusses how Agile software development impacts Code Quality. Part of this literature review was conducted before the Grounded Theory and a part was done after. We combined all literature reviews and put it first in the report to give the reader all the information that is required.

Although multiple perspectives on code quality exist we focus on the developers' perspectives. For example, an end-user might define code quality as the absence of bugs, but developers might see code quality as the ease with which the code is to be read or extended.

## 2.1 Agile Software Development and the Scrum Framework

Agile software development is an umbrella term for the development of software that follows the values of the Manifesto for Agile Software Development [8]. The main characteristic of Agile software development is that the team is able to respond to rapidly changing or uncertain requirements in complex projects.

Although there are multiple methodologies and frameworks for Agile software development, we only focused on Scrum because this framework is mostly used in companies around the globe [24].

**All information originates directly from the Scrum guide, last modified November 2017** [56], which is the official Scrum body of knowledge.

### 2.1.1 The Rules of Scrum

Scrum provides a lightweight framework that is simple to understand but difficult to master and has been used since the 1990s although officially documented in 2001 [54]. The framework is defined as follows:

> *"Scrum (n): A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value."* [56]

Scrum is not limited to software development and can be used for any kind of business. Scrum deals with complex problems that rapidly change. It focuses on small self-organising teams that are highly flexible and adaptive by employing an iterative approach rather than the traditional Waterfall approach.

Three pillars and five values uphold and are embodied by the Scrum Team. The first pillar is *Transparency*. It states that important aspects of the process must be made visible for the responsible people. A common standard should be agreed upon so that everyone has a common understanding of what is seen. The second pillar defines *Inspection*. All artefacts must frequently be inspected by the Scrum Users. It should not be too frequent that the inspections disturb the work but sufficient to detect undesirable variances at an early stage. The final pillar, *Adaptation*, states that when an artifact is observed to have undesirable variances and will result in a product that is unacceptable, the artifact must be adjusted as soon as possible. The five values are Commitment, Courage, Focus, Openness and Respect.

### 2.1.2 Roles and Responsibilities

A Scrum team has three roles i.e. Product Owner, Scrum Master and the Development Team. The teams are self-organising and cross-functional. Practices mentioned are discussed in section 2.1.3 (Sprint (Planning/Review/Retrospective/Backlog), Daily Scrum, Product Backlog, Definition of Done).

#### Product Owner

The Product Owner has the responsibility to maximise the value of the product that is being developed by the Development Team, and for managing the Product Backlog. The set of requirements as defined in the Product backlog are the solely requirements the Development Team works with. The Product Owner turns the issues of the Product Backlog into features to be developed.

#### Scrum Master

The Scrum Master is responsible seeing that the Development Team and Product Owner understand and correctly apply Scrum, and that the project is progressing as planned. The Scrum Master also helps in the interaction between external parties and the Scrum Team and decides which interactions are helpful and which are not. The interactions are guided and changed if required by the Scrum Master to maximise the value created by the Scrum Team.

#### Development Team

The Development Team consists of multiple individuals that develop the product. At the end of each Sprint, the team delivers a potentially releasable increment of the product. The team works on the assigned tasks from the Product Backlog during the Sprint.

The size of the Development Team is between 3 to 9 members. Fewer members put constraints on the amount of work that can be done in a Sprint and more than 9 members require too much coordination. The Product Owner and Scrum Master are not included in the Development Team unless they work on items of the Sprint Backlog. Scrum has been proven to work in large organisations with over 800 employees, but the organisation structure has to be split up into multiple Scrum teams [40, 53].

### 2.1.3 Scrum Practices

Scrum defines artifacts as documents which represent work or value and provides Transparency and opportunities for Inspections and Adaptations.

#### Sprint

A Sprint is an increment of the system. It has a fixed duration of one month or less during which all tasks for the next potentially releasable product are developed. The project is split up into multiple Sprints which all have the same duration and take place at the same location.

Once a Sprint has begun there can be no changes that endanger completing the Sprint Goal, the quality of the product is only increased. While more is learned about the project, the Product Owner is notified after which the decision can be made to modify the Product Backlog.

If the Sprint Goal becomes obsolete the Product Owner is allowed to cancel a Sprint. If a Sprint is cancelled all the tasks that are done are reviewed and added to the product if it is releasable. The remaining Backlog items are returned to the Product Backlog. However, Sprint cancellations rarely happen due to the short duration of Sprints and the additional time replanning takes.

#### Sprint Planning

The tasks that have to be done during a Sprint are defined in the Sprint Planning. At the beginning of the Sprint, the Sprint Planning is created by the Scrum Team, and is a time-boxed event of 8 hours for monthly Sprints and shorter for shorter Sprints (e.g. 4 hours for 2-week Sprints). Tasks contained in the Sprint Planning have a maximum effort of eight hours. If a task takes longer it should be split up into smaller tasks.

The Sprint Planning is meant to determine what can be delivered in the coming increment and what work needs to be done to deliver this. The Product Owner defines the objective of the Sprint and which items of the Product Backlog need to be finished to achieve the goal. When deciding how much work can be done, the team takes input from the Product Backlog, the previous Iteration, the projected capacity of the Development Team, and the past performance of the Development Team. The Development Team is responsible for selecting the number of

items selected from the Product Backlog. During the Sprint Planning, the Sprint Goal is defined. This is the objective that will be met in the coming Sprint.

When the items of the Product Backlog have been selected, the Development Team decides how this functionality can be put into a Done product increment. The plan to deliver this and the selected items from the Product Backlog are called the Sprint Backlog. The Development Team is able to explain to the Product Owner and the Scrum Master how they will accomplish their Sprint Goal.

#### Daily Scrum

Every day the Development Team holds a Daily Scrum meeting which is a 15-minute time-boxed event and is held at the same place and time each day. In the meeting, the Development Team plans the work that will be done in the next 24 hours.

The Daily Scrum is used to inspect the made progress towards the goal and inspect the progress towards the work that will be done next. After each Daily Scrum, the Development Team understands how to self-organise to accomplish the Sprint Goal and finish the expected increment.

The Daily Scrum has a maximum duration of 15 minutes, which is enforced by the Scrum Master. If more discussion is required the team can meet immediately after the Daily Scrum. The Scrum Master also makes sure that the Development Team is not interrupted during the meeting.

#### Sprint Review

A Sprint Review is held at the end of each Sprint. The increment is inspected and the Product Backlog is changed if needed. During the Sprint Review, the Scrum Team and stakeholders involved look back at what was done in the Sprint and what can be improved to optimise value.

The Sprint Review takes four hours at most for Sprints that have a duration of a month and shorter for shorter Sprints. The Scrum Master is responsible for the fact that it does not take longer than required and that all attendees understand the purpose of the meeting.

#### Sprint Retrospective

After the Sprint Review, the Scrum Team holds a Sprint Retrospective. While the Sprint Review was focused on the product, the Sprint Retrospective focuses on the Sprint Team itself.

The Sprint Retrospective takes three hours at most for a Sprint that takes a month, and shorter if the Sprint is shorter. The Scrum Master has the responsibility that it does not take longer than required and that all attendees understand the purpose of the meeting.

The Sprint Retrospective has three purposes. The first is to inspect the quality of the last Sprint with regards to people, relationships, processes and tools. Secondly, to find items that went well and how to improve it even further. Finally, a plan is created for implementing the improvements to enhance the Scrum team's way of working. At the end of the meeting, the Scrum Team has identified possible improvements and defined a plan that makes sure it is implemented in the next Sprint.

### Product Backlog

The Product Backlog is one of the artifacts of Scrum. It is a prioritised list of everything that is required to be present in the final product and prioritised by the importance of the items. The Product Owner is solely responsible for keeping the Product Backlog. However, the Product Owner can delegate tasks to the Scrum Team if needed. The list of the Product Backlog defines the requirements that are currently known. Therefore, the Product Backlog cannot be finished while work has to be completed.

Items in the Product Backlog can be features, functions, requirements, enhancements, and fixes. An item can have multiple attributes e.g. description, order, estimate and value. Items can also include an attribute to define when it is done.

There is one Product Backlog. If there are multiple teams working on the same project they work on the same Product Backlog but on different items. An attribute can be added to the items in the Product Backlog to allow the items to be grouped for a better overview.

At the end of each Sprint, the Product Owner can monitor the progress towards the goals by tracking how much work is left after the Sprint compared to the amount of work before the Sprint.

### Sprint Backlog

The Sprint Backlog is the second artifact and contains items that are selected from the Product Backlog, and a plan to deliver the Increment which realises the Sprint Goal. The Sprint Backlog makes time predictions about the functionalities that can be delivered in the next Increment and the amount of work that is needed to deliver it. It also includes at least one of the identified improvements from the Sprint Retrospective.

The Development Team constantly updates the Sprint Backlog with the estimated remaining work or removes items from the backlog that are no longer required. The Development Team is the only party that is allowed to modify the Sprint Backlog. The Sprint Backlog is highly visible and allows a real-time indication of the amount of work that is remaining. A burn-down or burn-up chart shows if the team is progressing as expected and can be used to further improve the transparency of the remaining work.

The increment that is to be delivered is the sum of all Product Backlog items that will be completed during the next Sprint and of all the previous Sprints. Each increment must be in a Done state and therefore be in a usable condition, ready to be released.

### Definition of Done

The term Done has been used multiple times. What Done means varies per Scrum Team but it is important that all Scrum Team members share the same meaning of what it means for something to be Done. Only when something is Done it is considered to be completed and ready for production. The common understanding of Done is stated in the Definition of Done, which is a living document.

## 2.2 Code Quality Standards

At the beginning of a project, a Development Team can create a list of coding rules which they should follow to ensure the quality of the product. In the Scrum framework, a *Definition of Done* is a living product which ensures that the integrated features are of high quality. Ensuring high-quality software is an important value for clients [57].

Using these rules the developers can ensure consistency and quality of their product. Knowledge and experience play an important role in improving this further. Developers can also participate in discussions or do code reviews to empower each other. However, the meaning of high-quality software is different for developers [49, 57]. What one developer values in terms of code quality may be different than what his peers value.

We have used the word quality six times in this section without the need of giving a definition. Everybody has a basic understanding of what quality means but it is difficult to give a proper definition. The Cambridge dictionary defines quality as:

> *"of a high standard."*[1]

When we look at the definition of code quality we see different ones in the literature, or literature which only uses the term code quality without giving a definition [13, 33, 23]. However, if there is no agreement on a single definition for code quality it cannot be measured properly. As a result, if we cannot measure it, we do not have enough knowledge on the topic. To quote William Thompson:

> *"I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be."* (William Thompson, 1883)

Although this quote is about physical science it illustrates one of the challenges when discussing code quality. Many existing definitions of code quality exist and discuss how software is being built best.

The Scrum framework provides a *Definition of Done* which ensures the quality of the product. However, this is a definition made by the Development Team and does not have to contain any rules related to Software Quality. This makes it difficult to define and measure the code quality of the product delivered, or what code quality means in this context.

To give a meaning to code quality we looked at companies that measure it. Since they are able to measure it, they know what they are talking about, according to William Thompson. This allows us to give meaning to the term code quality and provides knowledge on how to determine it in software projects.

[1] `https://dictionary.cambridge.org/dictionary/english/quality`

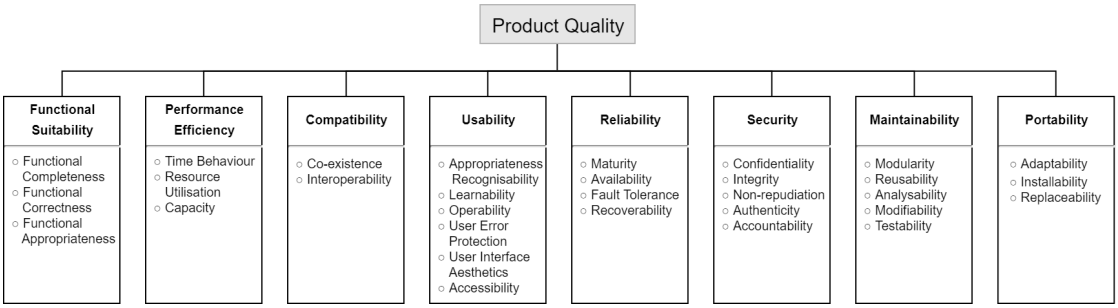


Figure 2.1: Characteristics and the related subcharacteristics of product quality properties as defined by ISO/IEC 25010.

ISO standards exist which define code quality. This standard is commonly used in literature and provides common ground we can use (e.g. [36, 45, 50]). Moreover, there are companies (e.g. CISQ[2] and Software Improvement Group[3]) that offer services to measure code quality evaluations. They use measures that directly follow from these standards. The Consortium for IT Software Quality (CISQ) is one of the prominent parties in defining how to automatically measure code quality.

### 2.2.1 ISO/IEC 25010:2011

ISO standards define requirements, specifications, guidelines or characteristics that can be used to ensure that materials, products, processes and services are fit for their purpose[4].

ISO/IEC 9126 was the ISO standard for the evaluation of code quality and was replaced by ISO/IEC 25000 in 2005, which was followed by ISO/IEC 25010 in 2011, and is part of the SQuaRE series of International Standards[5].

The standard defines a model which categorises the Product Quality based on eight different characteristics (i.e. functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability). Each characteristic is composed of a set of related subcharacteristics which is displayed in Figure 2.1. The full descriptions of the (sub)characteristics are defined in one sentence and can be found in footnote 5.

[2] http://it-cisq.org/
[3] https://www.sig.eu/
[4] https://www.iso.org/standards.html
[5] http://iso25000.com/index.php/en/iso-25000-standards/ISO/IEC25010:2011

## 2.3 Code Quality Measurements

The (sub)characteristics of ISO/IEC 25010:2011 do not provide a description of how to accurately measure them. Moreover, not all characteristics can be measured automatically/statically e.g. Functional Suitability. In this Section, we explored which characteristics can be measured. We linked them to Agile software development best practices to give an indication of which process to improve and how to get a higher score for a measurement for one of these characteristics in Chapter 7.

We studied multiple code quality analysis tools and which characteristics they measure. Many tools are available and there is no best tool. Therefore, we researched tools that were commonly mentioned on developer fora, blog posts or used in GitHub open source projects. We made sure that we covered enough tools to have a clear understanding in which characteristics of the ISO/IEC 25010:2011 standard can be measured automatically. An overview of the researched tools is given in Table 2.1.

Table 2.1: Static analysis tools/organisations and the characteristics they measure.

| **Tool/Organisation** | **Measurements** |
|---|---|
| CAST | Robustness, Efficiency, Security, Changeability, Transferability, Technical Debt, Size |
| CISQ | Reliability, Performance Efficiency, Security, Maintainability |
| Codebeat | Complexity, Duplications, Security |
| Coverity Scan | Security |
| Klocwork | Robustness, Security, Maintainability |
| SIG | Maintainability |
| SonarQube | Reliability, Security, Maintainability, Coverage, Duplications, Size, Complexity |
| Veracode Static Analysis | Security |

Although tools give different names to their measurements, we can identify four measurable ISO/IEC 25010:2011 characteristics from their documentations: Reliability, Performance Efficiency, Security and Maintainability. Additionally, Static Application Security Testing (SAST) is a functionality commonly implemented in tools. Furthermore, issues in these characteristics require work to be done to fix them. The amount of work that is needed is referred to as Technical Debt, which is also measured by multiple tools.

The measurements of the tools are based on predefined rules. These rules can be adjusted to suit the needs of the team. This complements the findings that everyone has a different perspective on code quality and teams can modify the tools to enforce their definition of code quality.

## 2.4 Code Quality in Agile Software Development

Literature shows that Agile software development has quality assurance in its processes [6, 28, 4, 10]. However, we found a minimal amount of studies that discuss how Agile software development influences code quality [35]. Therefore, we included unpublished literature that could be relevant. As a requirement, this literature had to be based on data from trusted sources, and be posted by an expert on sites related to the topic discussed.

Huo et al. [35] researched software quality assurance in the Waterfall method and discussed how this is done in Agile software development methods. Although it was concluded that Agile software development and Waterfall cannot be compared when looking at software quality, links to code quality and Agile software development processes are discussed. Unfortunately, the quality assurance processes of the Waterfall model were taken and with that perspective it was discussed how Agile software development processes are related to those processes. This can possibly have resulted in lacking practices due to a missing perspective from the Agile software development approach only.

Some Agile software development processes are given that can be used for software quality assurance. These are as follows: system metaphor, architectural spike, (on-site) customer feedback, refactoring, pair programming, stand-up meetings, cyclic redundancy check, simplify problems, pass 100% unit test, continuous integration, acceptance testing, code inspections, collective code ownership and using static code analysis tools.

One of the goals of this thesis is to establish Agile software development best practices to increase code quality metrics. This does not limit us to practices that are solely used in Agile software development. Therefore, the discussion about commonly used best practices described in a blog post by Dolan [26] can be useful later in the study. Five core categories are given in which practices can be applied for code quality assurance. These are Code, Team, Automation, Process and Testing. Useful Concrete practices that are given are: using static analysis tools, document code functionality, do reviews, pair programming, collective code ownership, collaborative meetings, automated testing of committed changes, automatic verification of committed changes, regression testing, minimal technical documentation, TDD, unit testing, integration testing, use coverage tools, automated GUI testing.

Dolan [26] and a forum on Stack Exchange[6] mention the balance between quality and velocity in Agile software development. It is discussed that Agile software development does not necessarily impact code quality, but that it has some built-in quality assurance.

[6] https://sqa.stackexchange.com/questions/25897/will-implementing-Agile-generally-improve-code-quality

The limited amount of related literature makes any claims about the relation between code quality and Agile software development disputable. There is a limited amount of organisations that work completely Agile and even then the code quality depends on the implemented Agile processes and how they are followed [53]. Since Agile software development is mostly in a state that is not fully mature, organisation use different practices that work best with the current state of their Agile software development implementation. Although different solutions are given, we observed that Agile software development processes do impact code quality, but the reasons behind it are not properly researched or unpublished. However, we have discussed useful literature which can be used to complement our results.

We conclude this section with an overview of discussed Agile software development best practices that impact the software in Table 2.2. The overview can be used later in the research to establish which Agile software development practices impact code quality and how they should be implemented to increase the code quality.

Table 2.2: Agile software development best practices and the source which discusses their usefulness.

| **Best Practice** | **Reference** |
|---|---|
| Acceptance Testing | [26], [35] |
| Code Reviews | [26], [35], [37] |
| Collective Code Ownership | [26], [35] |
| Continuous Integration | [25], [26], [35] |
| Customer Feedback | [4], [6], [35], [47], [68] |
| Cyclic Redundancy Check | [26], [35], [37] |
| Frequent Feedback | [37], [41],[47], [68] |
| Peer Programming | [4], [6], [26], [35], [47] |
| Reduced Documentation | [4], [26] |
| Refactoring | [4], [6], [35], [37], [47] |
| Regression Testing | [25], [26] |
| Stand up meetings | [25], [35] |
| Test Driven Development (TDD) | [4], [6], [26], [37] |
| Unit Testing | [25], [26], [35], [37], [47] |
| Use Static Code Analysis Tools | [26], [35], [37] |

# Chapter 3

# Grounded Theory

Grounded Theory (GT) is a methodology to do qualitative research where researchers systematically generate theory from data [30]. The idea was first coined by Glaser in 1967 and provides a means to generate theory rather than validate existing theory. In the book of Bryant and Charmaz [14], Grounded Theory is described as a contested concept. From this dispute, three main streams are acknowledged [3]: Classic or Glaserian GT, Straussian GT, and Constructivist GT.

Grounded Theory focuses on any kinds of data (e.g. interviews, transcript, documents) from which the researcher observes what is going on. Theories that are created can be tested by observing new and old data. This way the researcher inductively verifies new thoughts until a theory is defined that is grounded.

The constant collection and analysis of data is a core feature in doing Grounded Theory research. However, the first version of Grounded Theory defined many ambiguous procedures how to properly do Grounded Theory, which caused disagreements. As a result, multiple researchers diverged in the precise definition of Grounded Theory leading to three different versions as discussed by Strauss [62] and Adolph et al. [3].

We discussed that Agile software development impacts code quality but that it is not explained why. Grounded Theory gives us a means to find this theory.

## 3.1 Fundamentals of Grounded Theory

Despite the split in Grounded Theory methodologies, fundamental concepts exist which are shared between the different versions. Stol et al. [59] performed an extensive empirical study on studies that claim to use Grounded Theory. In their definition they describe core features of Grounded Theory. We provide an overview of their findings because they give an overview of what should be rigorously done when performing any Grounded Theory research:

- **Limit Exposure to Literature.** Instead of doing a comprehensive literature survey at the beginning of the study, researchers should limit the exposure to it to keep an open-minded look on the subject [31, 61]. If the researcher is exposed too much, a bias can develop from which theories and ideas emerge, which contradicts the goal of Grounded Theory.
- **Treat Everything as Data.** Everything is data and can be used in research [31, 59, 32].

- **Immediate and Continuous Data Analysis.** The researcher begins analysing data when obtained. The analysis acts as the basis for the next data that will be obtained [30, 16].

- **Theoretical Sampling.** Theoretical Sampling is performed after the analysis of data to fill gaps in the theory or to further explore concepts [30, 16, 61].

- **Theoretical Sensitivity.** Theoretical Sensitivity is the ability of the researcher to make concepts and connections between them. Creativity is an important role in this process [32, 61].

- **Coding.** During the Coding process the researcher constructs analytical codes and infers theoretical categories from data [30].

- **Memoing.** The researcher writes Memos to write down thoughts. These Memos help the researcher to structure data and to construct categories and their relations. If the stage of Memoing is skipped, the researcher is not following Grounded Theory [32].

- **Constant Comparison.** From the beginning of the study the researcher constantly compares data, memos, codes and categories with each other and with new data [30, 60, 16].

- **Memo Sorting.** During the study the researcher must continuously go through all memos and theories to find a core category. Similar to Memoing, Memo Sorting is critical in the procedures of following Grounded Theory [61, 32].

- **Theoretical Saturation.** Theoretical Saturation is the point where new data no longer trigger revisions or modifications of the theory [30]. When Theoretical Saturation is reached the researcher can stop collecting and analysing data and a resulting theory has been defined that is grounded.

## 3.2 Straussian GT

We decided to follow the Straussian GT. The well-defined procedures and rules make it easier to apply for new researchers in the field of qualitative research [20, 60, 19]. It also allows, but does not require, a definition of Research Questions before the start of the study. Moreover, a short Literature Study is allowed at the beginning of the study. Following the Straussian GT we can propose a general research question which we can use as basic guidance for the study, and perform a basic literature survey to get familiar with the basics of the Scrum methodology and code quality which has already been discussed.

### 3.2.1 Procedures

We used the published guidance of Strauss and Corbin [61] for our research. The publication defines 11 procedures one should follow to rigorously do Straussian GT. We provide a short summary of each procedure.

**Data Collection and Analysis are Interrelated Processes.** This procedure is similar to the core feature *Immediate and Continuous Data Analysis*. Strauss and Corbin [61] define a stricter policy where the researcher collects data, analyses those pieces of data and repeats that process. Hence, the researcher must complete the analysis before collecting new data.

**Concepts are the Basic Units of Analysis.** The researcher works with concepts of data. The concepts serve as the basic units for theory, rather than raw data. As the analysis continues the concepts become numerous and more abstract.

**Categories Must be Developed and Related.** Concepts that relate to similar phenomena can be grouped to form categories. Categories are more abstract than the concepts they represent and should be compared to highlight similarities and dissimilarities which in turn is used to produce lower level concepts.

**Sampling in Grounded Theory Proceeds on Theoretical Grounds.** The researcher draws samples in terms of concepts, their properties, dimensions, and variations. As such, the researcher samples incidents rather than specific individuals. The goal of theoretical sampling is that representativeness and consistency are achieved by building theoretical explanations of phenomena of the study.

**Analysis Makes use of Constant Comparisons.** This procedure is identical to the fundamental *Constant Comparison* of Grounded Theory.

**Patterns and Variations Must be Accounted for.** If a variation of the pattern emerges the researcher should account for it to give data an orderly basis.

**Process Must be Build Into the Theory.** Strauss and Corbin [60] give two meanings to process:

> *"Process analysis can mean breaking a phenomenon down into stages, phases, or steps. Process may also denote purposeful action/interaction that is not necessarily progressive but changes in response to prevailing conditions."*

**8. Writing Theoretical Memos is an Integral Part of Doing Grounded Theory.** This procedure is the combination of the core features: *Memoing* and *Memo Sorting*.

**9. Hypotheses About Relationships among Categories Should Be Developed and Verified as Much as Possible During the Research Process.** Hypotheses about relations among categories should be verified and adjusted as needed. As soon as a new hypothesis emerges the researcher takes it into the field and finds data related to it.

**10. A Grounded Theorist Need Not Work Alone.** Concepts are subject to interpretation. Discussing concepts and their relationships with colleagues can lead to new insights and increased *Theoretical Sensitivity* of the researcher, and enables collaborative analysis.

**11. Broader Structural Conditions Must Be Analysed, However Microscopic the Research.** The researcher must not be restricted by concepts that are immediately related to the phenomenon but should also consider broader conditions (e.g. cultural, political, social).

### 3.2.2 Coding

Coding is one of the fundamentals of Grounded Theory during which the researcher attaches labels to data that are being processed. Strauss and Corbin [61] define three basic types of coding that all have to be done: Open, Axial and Selective.

**Open Coding.** Open Coding is the process where data are broken down into codes and categories. Incidents are labelled with concepts which are compared and grouped together to form categories and subcategories. Once the categories are identified they become the basis for theoretical sampling. Open Coding can be done line by line, sentence by sentence or relating to the entire document.

**Axial Coding.** Axial Coding is used to relate categories to their subcategories and relationships are tested against data. All hypothetical relationships are repeatedly tested against new data until verified or contradicted. Once a hypothesis is compared to data repeatedly, it is properly tested to the issue of variation and conditions [61]. This phase has similarities to the *Theoretical Coding* phase described by Glaser [30].

**Selective Coding.** Selective Coding is the process where all categories are unified around one core category. This process is commonly executed at the end of the study. The core category is the central phenomenon of the research and answers questions such as: What is the main analytic idea presented in this research [61]? If a single core category cannot be committed to the research, the researcher should pick the one that captures the essence of the study.

# Chapter 4

# Experimental Design

In this Chapter we describe our research goal and the methodology used to achieve the goal. We used the work of Creswell's Exploratory Sequential-Qualitative first Mixed Methods approach [21]. We conduct an exploratory qualitative phase and further validate it using a quantitative phase. Following Straussian GT as defined by Strauss and Corbin [61], we perform an exploratory qualitative study to provide theories about the relation between Agile software development and code quality. During this phase semi-structured interviews are conducted to collect data. The Research Question are modified and new ones emerged during this phase. The results are tested in a quantitative phase using a questionnaire to cover the lack of validation of GT [31].

## 4.1 Goal

The goal of this thesis is to explore the relation between Agile software development and code quality and provide concrete solutions for developers on how they can improve their code quality. The literature survey showed that they are related but that it is not clear how or why. We aim to qualitatively define a theory about the relation. Following Straussian GT we have not defined initial Research Questions; they emerged as our theory grew. The final theory does not give a practical solution. Rather it provides insights which can then be used to continue with establishing a practical solution.

## 4.2 Qualitative Phase: Grounded Theory

From the procedures and guidelines of Straussian GT, we define a structured approach for conducting a Grounded Theory. Figure 4.1 gives an overview of the steps of our implementation which we will discuss in this Section.

Data was collected in the form of semi-structured interviews. We prepare 5-10 open questions and ask additional questions during the interview. These are questions about what, when, where, how, with what consequences, and under what conditions phenomena occur. The possibility exists that we cannot ask all our prepared questions because answers given are too elab-

orate. In that scenario we simply stop instead of rushing the interview or interrupting the interviewee.

The Data Collection via interviews is immediately followed by the Data Analysis. The analysis consists of Open Coding and Axial Coding. During the Open Coding we explored new concepts and their properties to define new categories. Axial Coding is used to relate categories to their sub-categories and explore their relations. Coding is done using the ATLAS.ti 8 tool for qualitative coding[1].

The Open Coding stage takes transcript of the interview and analyses it sentence by sentence. An interpretation of what the interviewee is saying is given from which conceptual codes are generated. We compared the concepts to the concepts of previous interviews to group incidents under categories. During the Open Coding stage we constantly wrote Memos to capture ideas from which we generate new questions to ask in upcoming interviews. The Categories and Subcategories deduced from the Open Coding are the main input for the Axial Coding.

Axial Coding takes the same transcript of the interview that were used in the Open Coding and defines categories and their subcategories that were explored while conducting the Open Coding. During this analysis we think about the conditions that might lead to our categories. Hypothetical relationships are proposed deductively and repeatedly verified against new interviews. This gives us conceptual linkages. As the research progresses, we no longer gain new information from the data. At this point, we have reached Theoretical Saturation and proceed to the Selective Coding phase.

The goal of the iterative process of Data Collection and Open/Axial Coding is to discover the Core Category of our study. The Core Category is the Category around which all other categories are grouped. The other categories will always have a relation to the core category as conditions, actions or interactional strategies, or as consequences. The first phase of Selective Coding is to determine the Core Category. Next, we look back at old data to identify new relations and categories that revolve around them. The Core Category represents the central phenomenon of the study and provides structure for the Grounded Theory.

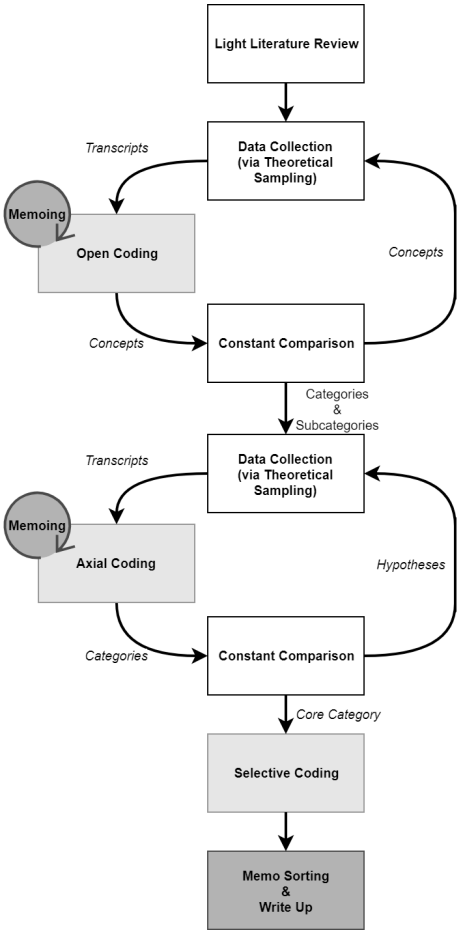


Figure 4.1: Overview of our implementation of the Straussian Grounded Theory Method.

[1] http://atlasti.com/

We finish the process with memo sorting.
During this phase we look back at all memos and group memos that are related. Sorting the memos provides the theoretical outline of the theory. Memo sorting is used to structure fractured data.

### 4.2.1 Extending the GT Model

Grounded Theory generates a significant number of data which has to be constantly compared to new data. It can be troublesome for new researchers to find meaningful concepts or determine incidents that have to be explored further in the upcoming interview. To deal with the large amount of data we added an additional type of coding which we applied before any other type of coding. We define this coding as *Molecular Coding*. We gradually decreased the amount of molecular coding as we grew in processing data, and as the data became more and more linked, at which point the memoing phase was sufficient to generate hypotheses and ideas where to conduct the next interview.

During Molecular Coding, we took small pieces of texts or even single words from quotations, which we call components. We can link the components to each other based on the quotations of the participants. Sorting the components in a hierarchical order gives us two major advantages. Firstly, it shows areas where the number of components is dense or sparse. Both can indicate interesting topics to be considered in the next interviews. Secondly, it shows which components are responsible for the majority of the rest of the components. These can provide linked concepts that have a potential to be directly linked to the emerging core category.

For example, a participant was discussing what would be an ideal team culture to receive and process feedback from. From this we obtained the following quotation and components:

> **Quotation 1.**
> **Interview Quotation:** *"I think that it is important to feel safe in a team. That you do not have the feeling that when you make a mistake your throat will be cut but that you are dedicated to collaborate to find your mistakes and those of others."* - P5, Agile Coach / Lead Developer / Scrum Master
>
> **Components:** Feeling safe in the team, Be allowed to make mistakes, Collaborate, Feedback, Team Culture.

Figure 4.2 shows how we arranged the components in a sorted graph and how the participant linked them. In larger sorted graphs we would have more significant components on the right. We provide the network of components and categories of our first two interviews in Appendix B. It is important to understand that we sorted this graph top-down instead of left-right to obtain a better overview. Thus, our most significant component can be found at the top of the network. Although the components are small, one can see that *Team Growth* is at the top. Consequently, *Team Growth* was included in the third interview.

The second addition to the Grounded Theory we made is a verification phase. Grounded Theory relies on the interpretation of the researcher. When the theory was defined we returned to our participants with the results, to verify if they agreed. If multiple participants disagree or

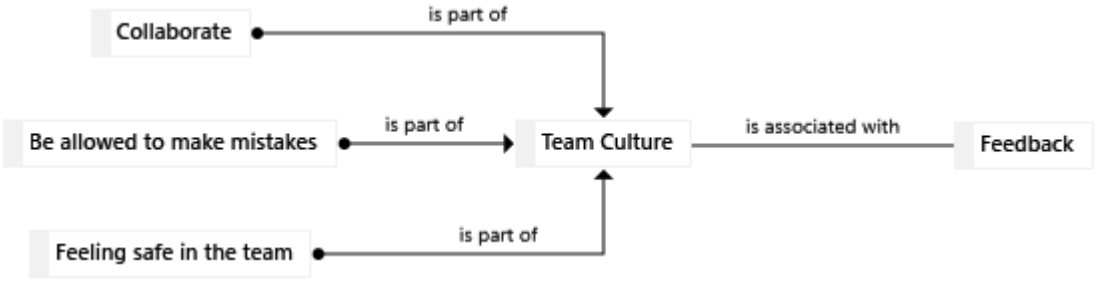


Figure 4.2: Example of components and their linkages.

provide similar new insights we can adjust the theory where necessary, reducing the chances of misinterpretations in the theory. This step is used to verify interpretations.

The final additional we introduced is the quantitative phase. Grounded Theory lacks a proper verification phase, making it difficult to generalise its findings. The questionnaire in the quantitative phase was used to add a second verification step to the developed theory. This step is used to verify the theory's generalisability.

### 4.2.2 Participants

During the qualitative phase we conducted semi-structured interviews of approximately 40 minutes to one hour, either face to face or via Skype, and recorded the audio. The participants came from two companies, one of which is a multi-national organisation. Its employees work together (outsourced) with the customers on the product. Additionally, they maintain applications. During the implementation they adapt to the customer's way of working. This gives us the possibility to get multiple perspectives on the way of working with Scrum. Not all participants are full-time software developers, but they all work with source code or the Agile way of working in software development.

Before we conducted the interviews we obtained approval by the University's Human Ethics Committee to collect and process data. We ensured all data are anonymously and safely stored. Participants had to sign a consent form before they were able to participate. Participants are referred to as $P_i$. The consent form is added to Appendix A. The participants were asked to join in our research using personal emails or by being approached directly. A total of 20 participants were interviewed. 19 were done face-to-face and one via Skype. All interviews were carried out in Dutch. Therefore, the quotations that are given are loose translations.

Table 4.1 shows an overview of our participants, their positions, how long they have worked in an Agile software development setting, and the method used for their current project. In addition to the Scrum and Waterfall method, we added the Scrum / Agile software development method. The participants using this method indicated they used some form of Scrum but made significant changes to the framework to suit their needs.

The data collection via theoretical sampling is a continuous process which helps to decide what data to collect from the next interviews based on the emerging theory. We wanted to have as much perspective on the topic as possible. Thus we interviewed participants in different roles with varying experiences. Participants with a preferred role were approached when the emerging theory required their perspectives. It is worth noting that all participants either work in a Scrum, Scrum / Agile software development, or Waterfall method. We were unable to interview other Agile software development methodologies like Extreme Programming.

Table 4.1: Participants and Experience (P# = Participant Number, BA = Business Analyst, Dev = Developer, AC = Agile software development Coach, PO = Product Owner, SM = Scrum Master, ASD = Agile software Development)

| **P#** | **Position(s)** | **Experience (in years)** | **Method** |
|---|---|---|---|
| P1 | Coach / Dev | 4-6 | Scrum |
| P2 | Architect | >10 | N/A |
| P3 | BA / Dev | 1-3 | Waterfall |
| P4 | Dev | 4-6 | Scrum |
| P5 | AC / Lead Dev / SM | >10 | Scrum / ASD |
| P6 | Dev | 4-6 | Scrum / ASD |
| P7 | Architect | >10 | N/A |
| P8 | Architect / Dev | 7-10 | Scrum |
| P9 | BA / Dev | 4-6 | Scrum / ASD |
| P10 | Architect / Management | 1-3 | N/A |
| P11 | Dev | 1-3 | Scrum |
| P12 | Dev | <1 | Scrum |
| P13 | Coach / Dev | 4-6 | Scrum / ASD |
| P14 | Dev | <1 | Scrum |
| P15 | Dev | 1-3 | Scrum |
| P16 | AC / BA / PO | 1-3 | Scrum / ASD |
| P17 | Dev | None | Waterfall |
| P18 | AC / SM | 7-10 | Scrum |
| P19 | SM | 4-6 | Scrum |
| P20 | Tester | 4-6 | Scrum / ASD |

## 4.3 Quantitative Phase: Questionnaire

The questionnaire is used to obtain quantitative data to verify theories and relations between code quality and Agile software development. Therefore, the questionnaire contains closed questions with a Likert scale. The questionnaire was made available online through a Google Form which, similarly to any online platform, allowed us to target a large group of respondents.

Bradburn et al. [12] describes 18 steps in preparing a questionnaire. We used this as a general outline in developing our questionnaire. The first steps are dedicated to deciding what

is needed. As mentioned we want to further verify the generalisability of the results from the qualitative phase. From this phase we established four values needed to achieve Team Empowerment. Each value had a conclusion containing the relation between code quality and Agile software development. Thus, we take these conclusions and transform them into a set of questions. Our hypothesis were that the answers would not diverge significantly from the theory. Four core questions are given around which the questionnaire has been built:

1. Does pressure reduce code quality and what Agile software development principles enhance or decrease the amount of being under pressure?

2. What is the role of function-awareness in the process of writing high-quality code and how does Agile software development fit in as a non-functional or aid in the process of creating awareness?

3. What are the main factors of Agile software development in enhancing the code quality?

4. Does the structure of Agile software development enhance/decrease the possibility to write high-quality code and should/must you change the structure if needed?

These are the core questions of the questionnaire. With these in mind we defined questions around each theme. Additionally, relevant literature that used a questionnaire were studied to reformulate and add questions [63, 38, 58, 66]. The consecutive steps require the questions to be placed in correct order and format to create a draft questionnaire. We used the four values (Chapter 5) and principles as themes and divided the questions into multiple pages. Respondents would see approximately six questions per section which involve a single theme. At the end of these steps we had six sections and 36 questions. For the first five sections, respondents have five multiple choice options: strongly disagree, disagree, neutral, agree and strongly agree. Each option is assigned a value from 1-5 (Strongly Disagree = 1 and Strongly Agree = 5) for the analysis. These sections were concluded with an open question where participants could elaborate on some answers. The final section contains three open questions to obtain additional insights into advantages and disadvantages of Agile software development.

Once the draft questionnaire has a form that can be distributed it is tested against the researcher himself, friends or coworkers. The feedback allows the questionnaire to be revised by changing, adding or removing questions. This step is repeated as long as significant revisions are made. We used our university peers and direct connections to test the draft. With their feedback we created the final questionnaire, which is included in Appendix F. The questions were added to a Google Forms and the link was shared.

In the final steps the researcher analyses the results and reflects on the process to improve in making future questionnaires. We used spreadsheets for the analysis and to generate statistics. Additionally, Google Forms provides an overview of replies. The information how long a respondent worked with Agile software development and the methodologies are used to study possible correlations and to increase the variety of our researched focus group. We included three questions to the questionnaire which we used to group respondents in the analysis: what their experience in Agile software development is, the methodology a respondent used and if the respondent prefers Agile software development over the Waterfall model. The latter can be

used to compare results against the attitude towards Agile software development. However, the purpose of the questionnaire is to further verify the theory and is therefore quantitative and not qualitative. The statistics help in the verification process or can uncover additional topics for further research.

### 4.3.1 Data Analysis

Collected data was analysed in two ways. Firstly, we created a boxplot of the Likert scales to analyse the distributions. The goal of the analyses is to verify the generalisability of the results. Therefore, we decided that at least the majority of the respondents had to agree to a statement before it can be considered to be generalisable. The values of the Likert scale were mapped to a scale from 1 to 5 where strongly disagree maps to 1 and strongly agree to 5. Following the work of Allen and Seaman [5] we took the median to evaluate if the majority of the respondents agreed. A distribution with a median 3.5 indicates that at least half of the sample group agrees with a statement. Although the median was the core parameter in deciding if respondents agreed, we also used the mean since a median would not be affected if half of the respondents strongly disagreed and the other half agreed. Thus, the threshold of a median of 3.5 or above was complemented with a threshold of an average of 3.50 or above.

Secondly, replies that contained open answers were analysed individually to verify if the reasons why respondents (dis)agreed could be related to the model. Moreover, we used this analysis to investigate why respondents agreed or disagreed with each other on similar statements. Answers on open ended questions also allowed us to find reasons why parts of the theory could not be generalised and how the theory could be modified to agree with both the interview and questionnaire data.

### 4.3.2 Participants

To corroborate our interviews and add a verification step in the Grounded Theory process to answer RQ2, we built a survey to be filled out by a global sample of developers who work in an agile way. The goal of the survey was to determine the generalisability of the Grounded Theory, using a Likert scale to calculate how much a respondent agrees. The survey was built using the work of Bradburn et al. [12]. The created 36 questions were split up into demographics, questions related to the four main values of the theory and a reflective session of the respondent's opinion about agile software development. A total of 24 questions used a Likert scale. The survey was sent via Twitter and we approached organisations directly. Respondents had four weeks to fill in the survey.

74 respondents, originating from 11 countries, filled out the survey and had an average of 5.7 years in agile software development. Respondent indicated to have worked mainly with Scrum (94.5%), Extreme Programming (XP) (35.6%) and Kanban (28.8%).

# Chapter 5

# Qualitative Results

In this section we will discuss the results of the Grounded Theory study in a similar sequence of the methodology.

## 5.1 Coding Results

The first step of the data analysis was the Open and Axial Coding and they were applied after each interview. Additionally, Open and Axial Coding were preceded by our Molecular Coding during the first interviews. Components and codes established concepts, categories and subcategories which are used to structure the emerging theory.

Concepts and categories are theoretical explanations of phenomena of the study. Our memos were helpful in determining concepts. We give a quotation and components we extracted from it as an example in Quotation 2.

> **Quotation 2.**
> *"It all comes down to culture, the culture of the people of the team and the change in culture. I will wait until something is put on my desk before I look at it. Instead, we are responsible as a team for the assignment that is given. And we will work on it as a team. And if the tester happens to be sick and there is a lot of testing work that has to be done, we will do that testing work for the tester. We do not wait until the project manager sends a new tester. Team responsibility, that is linked to culture, culture of the people but also of the environment."* - P16, Product Owner
>
> **Components:** Commit to be responsible, Culture, Self-organising teams.

Quotation 2 shows relations between the coded three components: Commit to be responsible, Culture, Self-organising teams. At the point where we had reached theoretical saturation and established a core category, we had a total of 30 concepts. A network of the concepts and their links is given in Figure 5.1. Additionally, Appendix C gives an overview of the concepts and their definitions.

The network shows two major branches, A and B. Branch A relates to incidents about code quality. During the analyses we found that all incidents that were related to code quality had

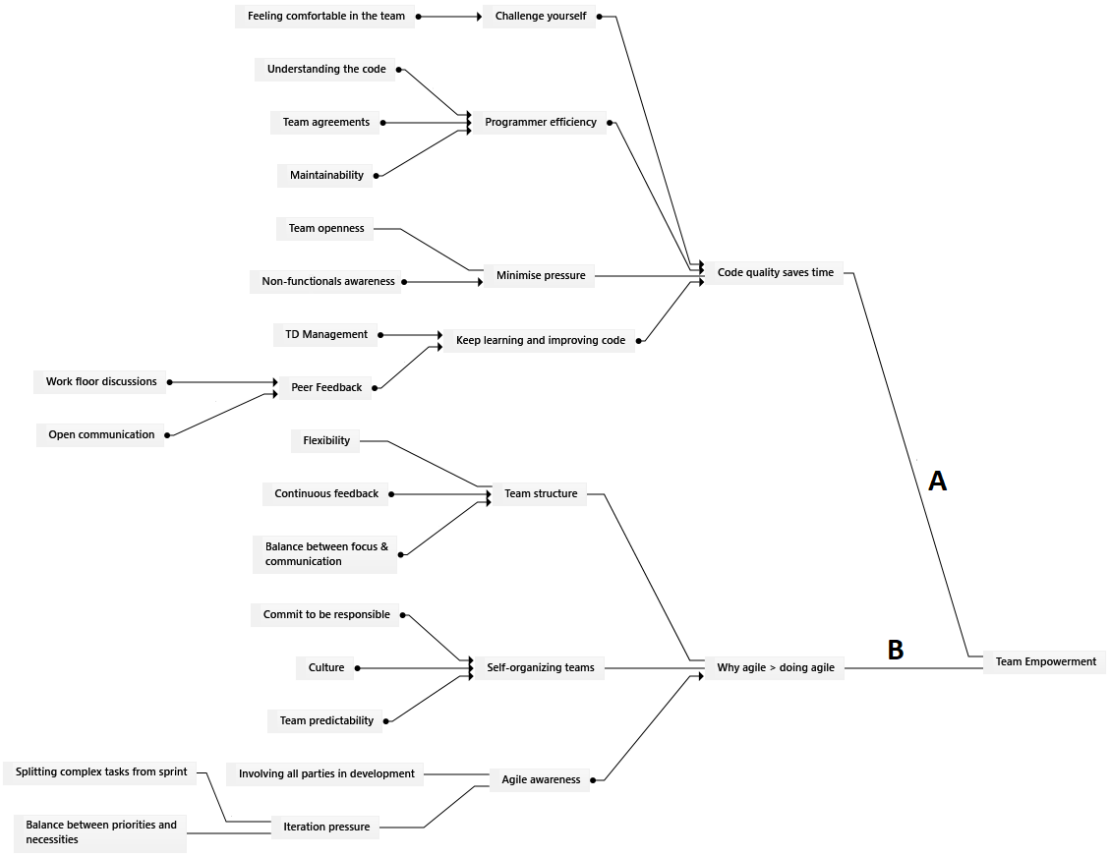


Figure 5.1: Resulting concepts and categories from the Open and Axial Coding. See Appendix C for the full definitions.

to do with reducing the development time. Branch B is built from concepts that relate to Agile software development. Similarly to code quality we found a single concept to explain the incidents; Agile software development is not about following a framework or methodology, it is more important to understand why you should do Agile software development. These two incidents gave rise to the core category Team Empowerment.

**Core Category:** Team Empowerment

We found that Team Empowerment was a suitable candidate to be the core category of our research. It accounted for all the observed incidents and occurred frequently in the data during the selective coding. Team Empowerment means that the team is responsible for the delivery of the product and possesses everything needed to achieve that goal. From this definition and our data a theory emerged, explaining the link from code quality to Agile software development via team empowerment.

Thus, the main theme in our research about the relation between code quality and Agile software development, is Team Empowerment. More specifically, if you want to achieve a high code quality in an Agile software development project, team empowerment is the value you should focus on in your Agile software development processes.

## 5.2 Emerged Theory

The final step of Grounded Theory is to sort the memos and write up the emerged theory. To connect the data we linked the memos to our categories and their links. From these we shuffled memos around until we formed an outline of the theory which was organised. An additional diagram was sketched and linked as we moved the memos around which provided the foundation for our theory.

The final theory contains three abstraction layers and is built from elements that provide a link between code quality and Agile software development. It can be seen as a pyramid where the lower elements support higher ones.

*Team Empowerment* is the roof of the pyramid and is supported by four values. All values are required and are the main characteristics and values a team should possess to become empowered. The lower abstraction level conditions support their respective value. The lowest abstraction layer, practices, are concrete rules or ideas one can apply in their business to stimulate the conditions. The practices are determined by repetitive incidents from the interviews. Figure 5.2 shows the full overview of values, conditions and practices. Each component is followed by a number in superscript. This will be used to directly link survey questions to components.

In this section we will describe Team Empowerment, followed by the values and their lower abstraction layer and briefly explain how they relate to code quality and Agile software development. Additionally, we research related literature on some of the elements to add support to our claims and possibly fill in the gaps in the interviews. We will wrap up every subsection with a paragraph on how the value, its conditions and its practices form a relation between code quality and Agile software development.

### 5.2.1 Team Empowerment

**Team Empowerment** requires teams to be self-sufficient, have a corporate authority and take responsibility to enact their own decisions. *Team Empowerment* requires four values. Firstly, if a team is pressurised too much it cannot be empowered. This does not mean that a team should not have any pressure at all. It should be sufficient to motivate the team without the team members being overwhelmed, at which point they have to abandon their beliefs to deliver their work. Developers will rush their work and as a result, code will be less maintainable, contain less documentation and contain more bugs. Thus the team should work in a *Sustainable Pace*.

Secondly, *Awareness* is required for a team to be empowered. With awareness we do not imply that the team is aware of what individuals are doing. Management needs to acknowledge the team's responsibility and allow them to work as they think is best. Awareness involves that the team informs required parties about relevant processes and what they think is needed. But

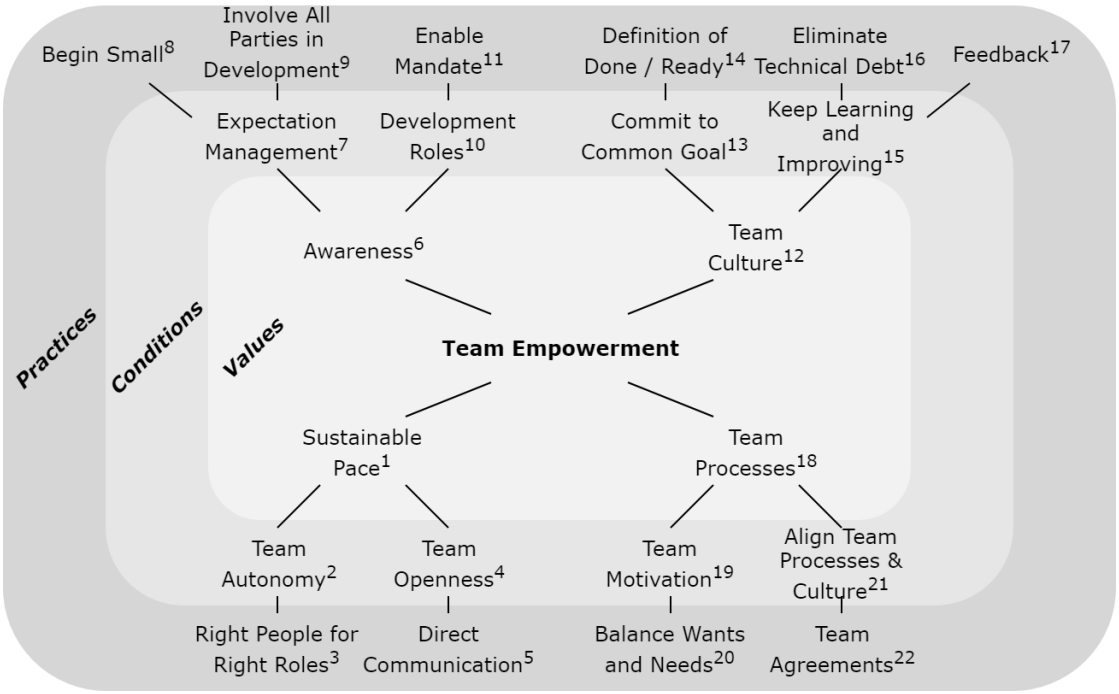


Figure 5.2: Values of achieving Team Empowerment, the conditions behind the values and practices mentioned on how to implement these conditions.

more important is the why-question behind processes. For example, having high-quality code takes time but is important because it will save time in the long term. It must be made clear that these non-functionals are part of the product and should be treated as such.

Thirdly, *Team Culture* determines how the team members interact and think. For a team to be empowered, the team should have a mindset where they take responsibility for their work. They take action themselves instead of waiting for tasks. Additionally, they want to keep learning and improving themselves and deliver the product as best as they can.

*Team Process* is the final value of *Team Empowerment*. There is no set of processes that works for everyone, but they help the team in understanding what is expected of them and what to expect of others. Clearly defined processes give guidance to the team on how to tackle tasks which improves their empowerment.

### 5.2.2 Sustainable Pace

The incident that has been mentioned most is that quality suffers under time pressure. Studies have similarly shown that pressure on developers decreases the quality of the code [7, 42, 28]. Moreover, Carmel and Agarwal [15] found that a methodology helps in decreasing the pressure, which also agrees with our *Team Processes* value. The *Team Culture* value impacts the ability to maintain a *Sustainable Pace* because an empowering culture is required for processes to work.

Maintaining a *Sustainable Pace* is built on top of 2 conditions. The first one is *Team Openness*. The team members are open to each other and the stakeholders. If they think they cannot meet the expectations they should inform the right parties. If there is an impediment they should let the right people know. Being open builds trust between parties. If you can be open you minimise pressure because you can indicate problems in time. To quote one of our participants:

> **Quotation 3.**
> *"I have seen some projects where the pressure became too high and you get the feeling that you cannot say everything anymore. [...] When you make a mistake you get penalised for having made the mistake instead of having worked together to prevent it or solve it."* - P5, Agile software development coach / Lead Developer / Scrum Master

Moe et al. [43] found that not being open in communication reduces the product quality. Participants mentioned that one of the easiest measures one can take to create an open environment is to enable *Direct Communication*. An example is a single table where the entire team works. This enables easy communication when an individual requires help, reduces the time to respond and minimises interpretation errors, but a Skype session can also be a form of *Direct Communication*. If communication is not direct it can take longer to solve problems and misinterpretations make it even last longer to resolve it (Quotation 4).

> **Quotation 4.**
> *"It is important that everyone is sitting together. Often you have a small question which you cannot post online and wait for an answer."* - P3, Business Analyst / Developer

The second condition, *Team Autonomy*, states that the team is able to make decisions by itself and solve problems without informing senior staff; it is about being independent. This also requires some backup from higher management. If higher management keeps interfering, you cannot achieve team autonomy. Autonomous teams take responsibility, are committed to deliver, and contain everything needed to achieve this. Participants claimed that all you need to achieve this is having all the right people on your team who are able to achieve all tasks. Bhasin [10] found that a high team autonomy combined with corporate responsibility is an important characteristic for Extreme Programming teams to be successful. However, no direct link with code quality was researched in this study.

To wrap up, pressure greatly reduces code quality. In order to maintain a *Sustainable Pace* one should aim for autonomous teams which have open communication, both internally and externally. Agile software development promotes open communication. To quote the Manifesto for Agile Software Development: "Customer collaboration over contract negotiation" [8]. The relation with open communication has also been verified by Pikkarainen et al. [46]. Additionally, Scrum, a framework implementing the Manifesto for Agile Software Development enhances *Direct Communication* and team autonomy by enforcing co-located teams and self-organising teams [55] which is a direct implementation of one of the twelve principles of the Manifesto for Agile Software Development: *The most efficient and effective method of conveying information*

*to and within a development team is face-to-face conversation.* [8]. Moreover, Agile software development fully promotes a sustainable working pace: *"Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely."* [8]. Thus the pressure on the team should be minimised to a pace where they can work in a *Sustainable Pace*.

### 5.2.3 Awareness

**Awareness** defines that every party is aware of relevant processes and why they have been implemented. When a customer wants a product he will be more interested in features and less in how it is built. The latter includes the code quality and the software development method. Moreover, developers might not be aware of the method their team is using and why, for example, they are using the Scrum framework. The team has to be aware of the value these tasks add, which was also found by Dybå and Dingsøyr [28], and Vidgen and Wang [65]. *Awareness* requires two conditions.

Firstly, *Development Roles*. Part of the awareness is that all involved parties are aware of the roles that exist, why they exist and what their responsibilities are. But also that parties are given the authority to be able to execute their role. This condition finds some if its roots in *Team Culture* and *Team Autonomy*. If you cannot guarantee that employees can execute their jobs as the processes demand, your processes will not work. Either your culture or structure has to change which will be discussed in the *Team Processes* value. Being able to give mandate helps (Quotations 5 and 6).

> **Quotation 5.**
> *"Maybe this belongs to culture, but a little acceptance of the role of the Product Owner, who is the boss of the Agile team. He has to be accepted and supported by higher management. [...] By giving a mandate, especially to the Product Owner."* - P16, Agile software development coach / Business Analyst / Product Owner

> **Quotation 6.**
> *You have a product owner that has mandate to make decisions. If the company does not, for example, have a culture to delegate, so that it cannot give mandate to that product owner. Then a part of your Agile software development process will not work because that product owner cannot make any decisions on the spot but always has to put decisions at his higher management of the organisation.* - P18, Agile software development coach / Scrum Master

The second condition of *Awareness* is *Expectation Management*. In Scrum the team rapidly delivers a minimal viable product. This is also a promise from the Manifesto for Agile Software Development, *"Working software over comprehensive documentation"* [8]. However, the fast delivery of features can give false expectations to the customer of the velocity the team is able to achieve. Not everything related to the final product is included in the first iterations of the product and including them will slow the visible velocity down (e.g. security). The client should be made aware of these tasks, the value and what this means for the delivery of the

system. Another aspect of *Expectation Management* is that the customer is made aware of what is expected of him and why. If the team follows the Scrum framework the customer must attend some of the events. The team should make the customer aware of why they are doing Scrum, how it benefits the process and why the client must attend the events. We took Scrum as an example scenario, but the same holds for other requirements that are not directly seen in the final product.

Two practices help in improving *Expectation Management*. The first practice is to *Involve All Parties in Development*. Stakeholders are not aware of the challenges in software development. Involving the stakeholders helps in creating awareness of the processes and provides opportunities at which the stakeholders can be made aware why some features take longer than others (e.g. caused by technologies) or other non-functionals. Moreover, something will inevitably go wrong during the project which can change the expectations and budget. Thus, the people responsible for the budget should be involved immediately to manage external expectations (Quotation 7). Another practice is to *Begin Small*. If a new methodology requires major changes in the organisation or culture of the business, do no try to implement it fully. Start small. For example, implement the change in a single team to experience it which allows the new change to be adjusted to suit the organisation better or vice versa. Small beginnings make the transition easier and help in creating understanding and awareness. For example, if you want to use Agile software development, do a proof of concept with one project first (Quotation 8).

**Quotation 7.**
*"It is also about psychology. When you have thought about something you have a positive attitude towards it. Thus, at the moment that you involve your stakeholders in the development process, regardless of the influence they possess, at the moment you include them in the process they are more likely to accept it and use it."* - P6, Developer.

**Quotation 8.**
*"The moment an organisation wants to do all their software development in an Agile way, do not implement it organisation-wide, but start slowly. A customer has to grow in the Agile software development process and do not expect from the customer that he does it perfectly from the first day."* - P18, Agile software development coach / Scrum Master

To wrap up, *Awareness* is important because, without it, your process is likely to fail which hurts the quality of the product and of the software. As we already mentioned, Agile software development can give false expectations by delivering rapidly. Therefore, the team should raise awareness that non-functionals are part of the product. The Scrum framework helps in raising awareness in two ways. Firstly, it offers clear responsibilities and descriptions of the roles in the development process. Secondly, the Sprint Review requires the stakeholders to be present which automatically involves them in the development process [55]. The Manifesto for Agile Software Development also contains a principle that stakeholders must be involved: *"Business people and developers must work together daily throughout the project."* [8]. The final link we

can establish between code quality and Agile software development is about the fact that Agile software development prioritises working software over documentation: *"Working software over comprehensive documentation"* [8]. From our interviews we learned that being able to understand the code is one of the two values of code quality. Documentation greatly helps in understanding the code. Therefore, the Agile way of working can hurt quality in the long term due to a lack of documentation (Quotation 9).

> **Quotation 9.**
> *"Yes, my previous client had that problem. They had been working for 3 years on a system and during those years they replaced all their developers. They either walked away or were sent away. If all your knowledge is part of the code then you have a serious operational problem."* - P19, Scrum Master

### 5.2.4 Team Culture

*Team Culture* determines how team members think and interact with each other. It is how they approach tasks and how they interact with obstacles. Processes support the team culture and vice versa [2]. Everything a team does and their reasons behind it is part of the culture. This does not necessarily have to be on a team level; it can also be on an individual level. However, we will focus on the team culture because, based on the interviews, we found that working as a team is better for the code quality than as a set of individuals. An empowered team takes initiative and decides what is best to do instead of waiting until the boss tells them what to do. Moreover, they always want to keep learning and improving themselves.

An empowering *Team Culture* is supported by two conditions: *Commit to Common Goal* and *Keep Learning and Improving*. The common goal gives guidance to the team on values they will follow and supports to embrace the culture or to gradually change the culture / processes (Quotation 10).

> **Quotation 10.**
> *"I think culture, the way of working cooperatively, in the sense of how do we work together? How do we do it together? It is extremely important to give a team a goal. And not a goal like build a project but to give a common goal like at the end of the project this is what we want to achieve, together."* - P18, Agile software development coach / Scrum Master

Having a common goal helps the team in thinking in the same way. Additionally, having a long-term goal helps in keeping the overview of the software. Instead of only working on tickets that have to be done, you look at how the functionality that you build fits in the bigger picture which in turn improves the code quality. A practice related to *Commit to Common Goal* is to introduce a *Definition of Done / Ready* for tasks. Because most of our interviewees were using Scrum this is most likely the reason why this was repeatedly mentioned. So there could be a alternative means to achieve a common goal.

Additionally, for an empowering team culture, the team should not fall behind on quality or make sure that if they do, they catch up. Moreover, developers should continuously work on improving themselves (Quotation 11). An example is the retrospective event from Scrum [55] during which the team inspects itself and creates a plan for improvement (Quotation 11).

**Quotation 11.**
*"So for example, that every week you sit down together to show a piece of code that was made during the previous week. Discuss what you have done, the problems that were encountered and how they were solved. That you review together to see, to check if it can be improved, what went well. Not to criticise people, it is important not to do that. But particularly to be constructive, and that you grow and learn together."* - P6, Developer

Two practices were repeatedly mentioned. One of which is *Feedback*. The Scrum retrospective is an example of feedback. Feedback is a method developers can use to learn from other developers and to improve each other when they see something that is not built as was agreed upon or that can be improved. However, feedback can be painful. Therefore, the culture of the team should enable honest feedback (Quotation 12). Examples of feedback are code reviews, pull-based development. Work-floor discussions also provide opportunities to give feedback.

**Quotation 12.**
*"When I am writing code, I do not know exactly if it does what it should do. Therefore, it is always useful that another person looks at it because he looks at it from a different perspective. Additionally, he does not know exactly what has been written which allows him to spot mistakes more easily. As an author you are frequently blind to your own mistakes."* - P1, Coach / Developer

The second practice that was often mentioned was to implement some kind of Technical Debt Management which we translated to *Eliminate Technical Debt*. Although Technical Debt does not exist in an ideal world, it is likely that it occurs in a project. When a decision is made that introduces Technical Debt (e.g. caused by a deadline) the team must document it and time should be reserved to work on it (Quotation 13). Thus, a simple "to do" will not suffice, it must be estimated how long it takes to fix the Technical Debt and reserve time to work on it. Because time is reserved for it the Technical Debt is eliminated since it is accounted for.

**Quotation 13.**
*"You should always do Technical Debt. That is what makes the difference between quality that is good and bad. There are always situations where you have to compromise due to planning or an upcoming demo. At the moment that you say something like, it is working, but we should do this and this because it will be better. Then you should register that as Technical Debt. Keep some kind of backlog for it and make sure that the product owner, scrum master and the person responsible for the budget agree with it and that time is reserved to fix it."* - P4, Developer

The Agile way of working requires a shift in cultural values [17] in which you take responsibility as a team [28]. Agile software development supports *Keep Learning and Improving*: *"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."* [8]. Additionally, the Scrum framework provides feedback opportunities, enforces a Definition of Done, and defines a common sprint goal [55].

### 5.2.5 Team Processes

*Team Processes* define the way of working of the team. It guides the team in working together and empowering itself. *Team Processes* affect the quality and having them well defined helps in aligning them with the team its values [44].

In order to get *Team Processes* that enable the right behaviour for *Team Empowerment*, the processes should allow the team to be motivated and align the processes with the culture of the team. Following defined processes is important when building software. However, one should not hold onto processes that do not work (Quotations 14 and 15). Moreover, the processes and the culture must align for them to work [17, 29]. Both the culture and the processes can be changed to achieve alignment.

> **Quotation 14.**
> *"Often you stick to a structure because you have to keep the structure without someone checking what works in the team. Maybe that is why Agile possibly breaks more than it should. But on the other hand, you must have some kind of process to build software."* - P9, Business Analyst / Developer

> **Quotation 15.**
> *"You use what you need and remove unnecessary things. It is in line with the original tooling. You keep tools that you need and everything that you do not need you remove."* - P20, Tester

Establishing *Team Agreements* on the processes and how they work is a practice one can use to help to align the culture and processes. However, if parts of the processes do not work the team should be able to remove them. An important aspect about *Team Agreements* is how the requirements are established. This was a topic where interviewees had conflicting opinions. How you establish the primary requirements is partly determined by the organisation culture. Some cultures require more certainty than others. There is not a best way to establish requirements, but it is important that the team agrees on requirements and that they agree with the team's structure and culture (Quotations 16 and 17). *Team Agreements* also define the coding guidelines or what happens when someone is too late for a meeting. It covers everything that the team agreed upon, either verbally or in written text.

> **Quotation 16.**
> *"Scrum provides a nice setup. Here are 20 pages, if you follow that you should do fine, and it will. However, there comes a moment that you must say, we are doing it this way. You use the techniques but use what works for you."* - P14, Developer

**Quotation 17.**
*"They want to determine most of the requirements in advance. That can very well match with the company, with the contract you have. So yes, I think all customers attempt to implement Agile in their own way of working."* - P18, Agile software development coach / Scrum Master

The second condition related to *Team Processes*, *Team Motivation*, is the hardest condition to implement. Every team is unique and has different team members. How they remain motivated depends on the individuals, but having good management helps [37]. However, people should be given the time to work on something they enjoy (Quotation 18). You should be given enough time to remain motivated (Quotation 19). Thus, it is important to balance the work you want to do and the work that has to be done [42]. Additionally Jones [37] describes best practices for motivation and morale of technical staff. For example, offer training and educational opportunities or give awards for outstanding work.

**Quotation 18.**
*"Simply look around to see if everybody is comfortable. Does everybody have enough work to do? Work that suits him? A good indicator is if he still likes it. If someone does not like it anymore then something is wrong."* - P5, Agile software development coach / Lead Developer / Scrum Master

**Quotation 19.**
*"Maybe allow people to work on tasks they want. Because then you will get a better motivation."* - P3, Business Analyst / Developer

Agile software development does not define *Team Processes*. However, one of the Agile software development principles states that *"Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done."* [8]. We can see this in the Scrum framework which provides an environment with events, artifacts and roles to give rules and structure to the team [55]. Furthermore, Bhasin [10] concluded that built-in processes like the daily standups are a tool for code quality assurance. If the culture of the organisation does not align with the Agile software development framework, it is better to use a structure that lies closer to the culture of the team. It can gradually make a transition where you have a culture that could adapt to an Agile way of working. Therefore, it is more important that you understand why you should use a methodology than simply following the rules. Because if there are major conflicts with the culture of the organisation, the processes will not work.

## 5.3 Verifying the Results

The questionnaire is used as a quantitative phase to further verify the results. However, the created theory was build using our interpretation of the quotations. To counter the threat that interpretations were incorrect or incomplete we returned to the participants. We selected six participants who were developers to discuss the created model. They were selected based on

different aspects. Firstly, these participants were all able to explain their ideas well. Since our model was built on quotations only, we wanted to discuss the model with different perspectives. Therefore, we selected participants who had major differences on some of the concepts of the theory.

We invited the six participants for another interview of an hour. These were structured interviews were we asked four questions. Firstly we handed Figure 5.2 to the participant and asked if the participant could explain the model to us. This step was done to improve unclear concepts. Secondly, we explained the model and asked if everything was clear, if something was incorrect or if something was missing. Thirdly, we asked if they could mention factors of a project where the code quality could have been better and how this relates to the model. Finally, a similar question was asked for projects with good code quality.

The goal of this verification step was to tweak the model. We wanted to make the concepts more clear and correct. For example, we had the concept of *Team Structure* which defines the processes of the team on a daily basis e.g. Scrum events. After discussing we decided that it was better to change this term to *Team Processes*. The second question was used to reduce the chance that we misinterpreted our participants. Using the four questions we tweaked the model to the form it was presented in this thesis. Moreover, the final two questions were used as ideas to make the model concrete and practical, which is one of the next steps in this thesis.

# Chapter 6

# Quantitative Results

The results from the Likert scales were used to verify the generalisability. These were combined with the results from the open questions to verify if the answers conform with the theory. The median and average were used in determining if the majority of the participants agreed or disagreed. Figure 6.1 gives an overview of whisker plots of the 24 questions that used Likert scales. The blue line indicated the 3.5 threshold. Additionally, Table 6.1 provides a link between questions and the superscript numbers of the components in the theory. Not all components of the Practices layer were directly linked to questions Moreover, question 24 asked if the respondent prefers agile over Waterfall and is therefore not directly linked to the theory.

Table 6.1: Link between the questions numbers (Appendix F) and the component numbers in the theory.

| **Question number** | 6 | 7 | 8 | 9 | 10 | 11 | 13 | 14 | 15 | 16 | 17 | 19 | 20 | 21 | 22 | 23 | 24 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Theory component** | 1 | 2 | 1 | 1 | 5 | 4 | 10 | 9 | 7 | 7 | 6 | 12 | 17 | 15 | 12 | 15 | 13 | 18 | 22 | 21 | 21 | 18 | 19 | - |

The results indicate that participants have not agreed significantly to question 30 (median: 3.5, mean: 3.41): "*I feel that agile software development increases my development speed*". The explanation that can be extracted from the open question was that agile software development does not increase individual developer speed, but it increases the speed at which the development team delivers what is required. For example:

> **Quotation 20.**
> *"It increases overall productivity, but not necessarily my own."* - Respondent 8

After the data from the interviews were revisited, component 18 (Team Processes) of the theory was modified to include that agile software development, if properly executed, should improve the speed of the Development team instead of the speed of individual developers.

Firstly, survey data were analysed individually to find further disagreements between the theory and the answers from the open questions. Respondents that agreed to a question may also have different opinions on why they agree than the ones described in the theory. 60 respondents added an explanation to their answers. Explanations why respondents disagreed were searched during the analysis of the individual responses. Most questions respondents disagreed with are covered by the theory. For example:

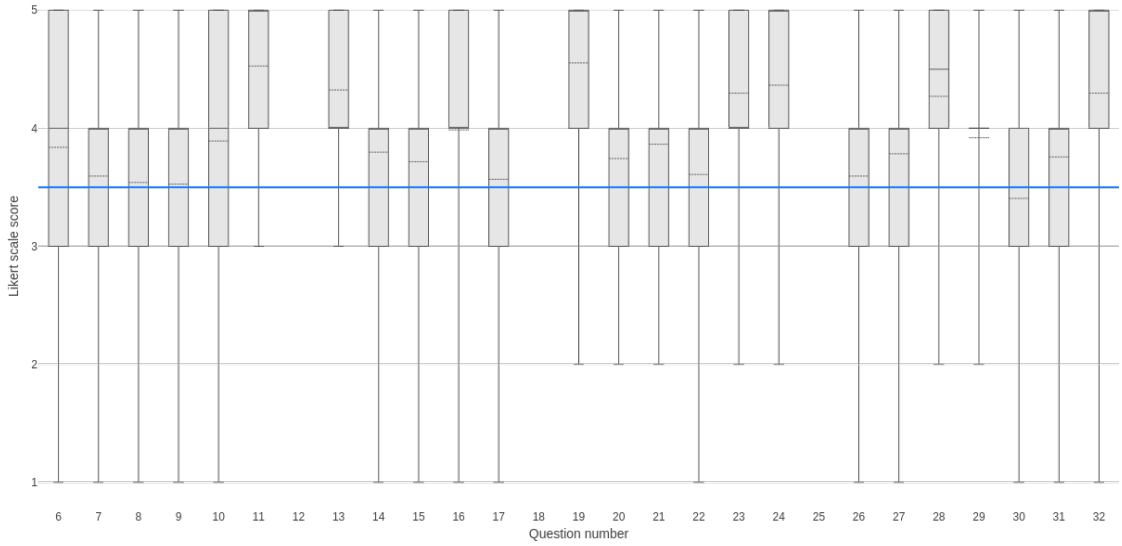


Figure 6.1: Whisker plots of Likert scales for each question, where 1 = strongly disagree and 5 = strongly agree (dotted line = mean, blue line = 3.5 threshold).

**Quotation 21.**
"*Disagree because PO always over plans the sprint, preventing us from completing it almost always.*"- Respondent 2

This quotation is covered in **Sustainable Pace** and **Right People for Right Roles**. Moreover, 28.3% of the respondents explicitly stated how influence from external parties caused them to disagree with a question (e.g."*Often the really important decisions are made at the management level outside the teams*" - Respondent 29). External influence conflicts with **Team Empowerment** since they do not have real corporate authority.

Following our threshold, respondents agreed to the other 22 questions related to the theory. Thus we can conclude that the theory can be generalised outside of the field in which the interviews were conducted. However, the combination of data from the interviews and survey can be combined to make additions to the theory. The data showed how external influence can break components of the theory which was covered in the qualitative results. However, two other aspects which have not been occurred in the interview data emerged. Firstly, 6.7% indicated that team size plays a role in building code of high quality. Secondly, 10% stressed how the Product Owner can break components of the theory.

For completeness we also review questions that have a median of 4.0 and a mean below 3.75 (i.e. 7, 8, 9, 15, 17, 22 and 26) and discuss answers of respondents that disagreed. We will explain why respondents disagreed and how this fits in the theory.

*Q7. I feel Agile software development allows the development team to operate more independently.* Two kinds of answers were given why respondents disagreed with this question. Firstly, the Product Owner can decrease the Independence of a team because he may enforce wrong decisions (Quotation 22). Secondly, external parties can limit the amount of Independence a team

has (Quotation 23). The theory accounts for both phenomena. The role of the Product Owner is described in component **Development Roles**. If the Product Owner ignores the developers, the role of the Product Owner is not applied correctly. The **Team Autonomy** component states that teams should be able to work independently. However, we were aware that this could be difficult to achieve. Therefore, answers like Quotation 23 can be categorised under *Align Team Processes & Culture*.

**Quotation 22.**
*"As a member of an agile team, you still have to be able to push back if e.g. the product owner wants features in the wrong order or if he/she wants to pressure you into finishing a feature earlier than you'd like."* - Respondent 59

**Quotation 23.**
*"Governance departments within an organization may impose constraints on the Independence of the development teams."* - Respondent 13

*Q8. Being part of an independent development team reduces unnecessary pressure.* Respondents that disagreed to Question 8 gave one motive. Respondents argued that pressure is not necessarily caused by team structure, but caused by culture, or communication with external parties (Quotation 24). We agree with this since it was already stated in the theory that components are not only directly related to linked components. A **Sustainable Pace** is indirectly influenced by **Team Culture** and **Awareness**, for example.

**Quotation 24.**
*"Being part of an independent development team can reduce pressure, but is not always the case. Stakeholders, project managers and even the Product Owner can stress the team by putting too much pressure on the team. This is especially true with teams that do (yet) not perform well."* - Respondent 14

*Q9. I feel Agile software development allows me to work in a sustainable pace.* Quotation 21 is an example of the main motive respondents gave when disagreeing to Question 9. This stresses the implication of the Product Owner on the **Sustainable Pace** component. The quotation is another example of how components from different trees influence each other since, in this motive, **Development Roles** influences **Sustainable Pace**.

*Q15. I feel (technical) decisions made by our team are respected and accepted by people outside of the development team.* One motive was given by respondents that disagreed to question 15. The Product Owner or management enforces its own decision (Quotation 25). This motive again shows how important it is that you have a good Product Owner.

**Quotation 25.**
*"Product owner: we need this feature. Team: That brings a lot of issues, doesn't fit our architecture. We can do it in another way. Product owner: customer needs this next week, do it anyways. Managers tend to side with the PO or customer. Software undergoes degrading because of these kinds of issues."* - Respondent 60

*Q17. Agile software development increases the efficiency of our development team*. Respondents that disagreed to Question 17 all gave a similar answer: it can be less efficient in the long term (Quotation 26). The component **Commit to Common Goal** states that you should have a common goal and keep an overview of the product as a whole instead of focusing on small parts that are done during a sprint.

**Quotation 26.**
*"Agile processes make it easy to only look at the short term, so if you only focus on that, you will become less efficient. It's important to keep your eyes on the long-term goals and designs. It's possible with Agile, but doesn't really feel built-in."* - Respondent 57

*Q22. I feel everyone is included in the decision-making processes of my development team.* The answers to Question 22 indicated two main reasons why respondents disagreed. Firstly, not all team members should be included in all parts of the decision-making process because that part might not be their expertise (Quotation 27). Secondly, not all team members engage in the decision-making processes because they are not experienced or articulate (Quotation 28). The data from the interviews agree with the first motive. Time is spent unnecessarily when everyone is involved in specific decisions. However, non-specific decisions should be discussed with the entire team. We changed this in the theory. From the interviews we also heard that not everyone is participating equally in the decision-making processes similar to the reasons respondents gave. However, it is important that these individuals **Keep Learning and Improving** so that they can participate more and stimulate them to actively join in the meetings.

**Quotation 27.**
*"Not everyone in the team should be involved in all decisions, i.e. a functional tester without software/architecture background shouldn't necessarily be involved in architecture."* - Respondent 64

**Quotation 28.**
*"As a Scrum Master in my current team I feel that not everyone is included in the decision making process. Often the junior developers do not engage in the discussion. But as a Scrum Master and coach, I think that they do need to be included. It also depends on the person itself."* - Respondent 12

*Q26. I think agile software development makes work more organised* Two respondents that disagreed to question 26 explained their reasons which were different. Therefore, no valid conclusions can be made from these two answers.

# Chapter 7

# Agile Software Development Best Practices for Improving Code Quality

This Chapter is dedicated to making the results from the Grounded Theory more practical. Although the model provides insights for piloting or modifying an existing project to increase code quality in Agile software development, the model is not complete and does not provide concrete solutions to improve code quality as measured by a code quality tool. Therefore, we go one step further and build on top of our theoretical model to provide insights for development teams on how they can increase their code quality through Team Empowerment.

The goal of this Chapter is to establish best practices that can be applied to Agile software development project. These practices must correspond with the results from the Grounded Theory since we are trying to improve code quality in Agile software development projects and not in different settings. Descriptions are given how these best practices should be implemented to enhance code quality. The data from the interviews are complemented with literature about best practices to link them to measurable code quality subcharacteristics: Reliability, Performance Efficiency, Security and Maintainability.

Developers who use a code quality analysis tool will get ratings on these characteristics or different ratings that are related to them. If one of these metrics is not as high as it should be or is getting worse over time, developers can use our results to find best practices to increase this metric. Although we do not recommend that the best practices should be blindly followed as described by us, they can be used for developers to find out which measures can be taken to increase code quality metrics. If a practice will not work in the team or organisation it should not be followed, or be changed properly before implementing.

## 7.1 Agile Software Development Best Practices

The best practices of the theory are based on 20 interviews. To increase the legitimacy and extend the list of best practices, we reviewed literature about Agile software development best practices and verified if they were applicable to our theory. Moreover, the results from the Grounded Theory allowed us to search literature more accurately, allowing us to find additional literature to obtain more best practices that are applicable to the model. To complement the best

practices we combined results from the literature survey and two sources that discuss software best practices of successful projects [37, 47]. If a best practice is given that fits in our theory we researched it further to verify if it is beneficial and which metrics are influenced by the practice.

The best practices are meant for developers or development teams to adjust their processes, allowing them to write better code. Although there are other best practices that can be applied outside the team, we have not included them since the resulting scope of the Grounded Theory was about team empowerment.

The combination of literature reviews and revisiting the interview data shows 3 additional best practices that apply to the model. Firstly, Continuous Integration which is a form of automated feedback to keep on learning and improving. Secondly, Refactoring the code, which is also part of the keep learning and improving component. Thirdly, Version Control Systems, which allow the team to implement features like design traceability and get automatic feedback. The Agile software development best practices from the literature survey are directly or indirectly included e.g. use static code analysis tools is combined with the Continuous Integration practice. We also considered including team size because it is often mentioned as a factor relating to code quality [48]. However, the team size depends on too many factors to convert it to a best practice and is best determined using experience [48]. Additionally, we also have not included the *begin small*, *right people for right roles* and *balance wants and needs* components of the theory in the best practices since the implementation is too different for each team. However, they are important when modifying the team processes to increase code quality and thus our theory should be understood before applying any of the best practices that are advised in this Chapter.

We determined the existence of nine best practices that can be used in Agile software development processes to enhance code quality:

1. **Continuous Integration.** Introduce Continuous Integration in your development pipeline to automatically run tests, check on your coding guidelines and practices and to verify if your code builds.

2. **Definition of Done.** Define rules when code is considered to be Done and ready to be merged to production. Include quality rules in the Definition of Done.

3. **Direct Communication.** Communication between team members is direct. Most communication should go via a face-to-face discussion or Skype/phone-calls. Communication where you have to wait for an answer should be minimal. This decreases the time developers are stuck and reduces misinterpretations.

4. **Eliminate Technical Debt.** All Technical Debt that is created should be accounted for. For example, if you have to take a shortcut to save some time, add the task that has to be done to properly implement that shortcut as Technical Debt and estimate how long it will take, instead of adding a simple TODO in comments. Take measures to be able to work on Technical Debt.

5. **Feedback.** Feedback can be between developers or on an individual level e.g. analysis tools. Enable developers to give feedback on each other's way of working and written

code. Introduce tools that help developers in following standards or to find possible flaws in the code.

6. **Involve All Parties in Development.** Parties that have an interest in the product should be involved in the development. Developers are not the end-users of the system and might not have all the information to perfectly make the product. Allowing parties involved to provide feedback gives developers a better picture of what has to be done, has gone wrong or could be done. It enables discussions between technical and non-technical staff which makes it easier to explain technical decisions.

7. **Refactoring.** If the code is not as good as it should be, take time to refactor some parts of the code that is frequently worked on. Not only will this save time in the future because developers can add features more easily, it also stimulates developers to write high-quality code in these parts.

8. **Team Agreements.** You can make agreements with coding standards or practices your team will use to enhance code quality. Implement measures to check that code satisfies the agreements.

9. **Version Control System.** A Version Control System gives another means of documentation. Design choices can more easily be tracked. It also provides a way to work faster together through branching, for example.

**You should decide what and how to change because we do not know where you are or where you are going. We do not know what you should do because we do not know what happens next. You should find out where you are, what you need and how you should adjust. The results of this Chapter should only be used to gain insights into how you can change.**

## 7.2 Relating Best Practices to Measurable ISO/IEC 25010:2011 Characteristics

Table 7.1 shows the established best practices and which ISO/IEC 25010:2011 metrics they can improve. Each best practice gives a general idea of how the processes of the team can be altered to improve quality scores. We do not impose any of the best practices. Teams should decide what can work for them and create a modification that is based on the idea provided. Entries are followed by references that have influenced the claim or support it.

The links between the best practices and ISO/IEC 25010:2011 metrics have been based on the interviews and existing literature. There are other useful best practices that might improve metrics than stated in the Table 7.1. For example, the Definition of Done can also contain rules about security metrics. However, from the data, these are uncommon scenarios and therefore not included in the table. Thus, the best practices can be used for other metrics if the team can apply them in that way. It is even encouraged because, thinking about why you do things is more important than following a predefined structure, which was covered in the Grounded Theory. Additionally, a poster was made to present the results to a business which can be found in Appendix G.

Table 7.1: Best practices and how they can increase ISO/IEC 25010:2011 metric measurements.

| **Best Practice** | **Maintainability** | **Security** | **Reliability** | **Performance Efficiency** |
|---|---|---|---|---|
| Continuous Integration | Use a static analysis tool that can measure maintainability and runs automated tests regularly [27, 40, 67]. | Use a static analysis tool that can measure security of the code which runs regularly [27, 47]. | Use a static analysis tool that can measure reliability of the code which runs regularly [27, 47]. | Use a static analysis tool that can measure efficiency (e.g. cyclomatic complexity) of the code which runs regularly [27]. |
| Definition of Done | Introduce code quality rules in the definition of done e.g. standards that must be followed, the requirement of two code reviews or minimal test coverage [22, 39]. | - | - | Define rules for complexity of methods, queries or the architecture of the database [22, 39]. |
| Direct Communication | Use direct communication to allow questions to be answered rapidly and accurately or to work together on a piece of code [37, 44, 47]. | - | Corner cases in which the software does not work are identified more easily when discussing them [44]. | - |

| Best Practice | Maintainability | Security | Reliability | Performance Efficiency |
|---|---|---|---|---|
| Eliminate Technical Debt | If TD is caused because something else was prioritised over maintainable code, include it in the TD backlog give developers time to work on it [39, 47]. | Temporary fixes that allow software to be usable but potentially introduce flaws, are put in the TD backlog and time is given to work on it [39, 47]. | - | A sub-optimal solution can be considered as a TD. Add the task that the algorithm has to be redesigned or add tricks from the platform have to be used to make it faster and be given time to work on it [37]. |
| Feedback | Automated/Developer feedback helps developers in coding better in the future. Code feedback helps in improving the maintainability of old code or code that is waiting to be merged, and indicate which parts of the code are not clear [1, 18, 47]. | Feedback on the code can identify vulnerabilities in the code that could have been easily overlooked by the person that wrote the code [1, 18]. | By having multiple developers review the code, problems that cause crashes can be identified earlier [18, 47]. | Feedback aids the problem-solving process, creating efficient solutions [1, 18]. |
| Involve All Parties in Development | Involving all parties can help in creating awareness why maintainability is important and increases the chances that the team will be given time to spend time on it [47]. | - | Gaining insights into the way in which the software is actually used helps developers in identifying problematic scenarios [64]. | - |

| Best Practice | Maintainability | Security | Reliability | Performance Efficiency |
|---|---|---|---|---|
| Refactoring | Take time to refactor code to be more maintainable because future code will also be more maintainable. High-quality code stimulates the development of high quality code [37, 40]. | - | - | - |
| Team Agreements | Similar to the Definition of Done, maintainability rules can be included in the team agreements. These can also be checked using an automated static analysis tool or using code reviews [22, 51]. | - | - | - |
| Version Control System | Version Control Systems provide effective change management control, allowing design choices to be tracked more easily or to revert to a more stable version [47, 67]. | - | - | - |

# Chapter 8

# Revisiting the Research Questions

The goal of this thesis is to explore the relations between code quality and Agile software development and establish concrete solutions to increase code quality in Agile software development. A qualitative phase was used as an explorative part, followed by a quantitative phase to verify the qualitative phase, which was extended in order to be able to be applied in a practical context. In this Chapter we will systematically revisit each research question and answer them based on the results.

**RQ1:** What is the main phenomenon that explains the relation between code quality and Agile software development and how does this explain it?

The qualitative phase uncovered team empowerment as a core relation between code quality and Agile software development. Team Empowerment means that the team is responsible for the delivery of the product and possesses everything needed to achieve that goal. Team empowerment is supported by 4 values, 8 conditions and 10 practices (Figure 5.2). These consist of conditions for achieving team empowerment in Agile projects and how these conditions can manifest in code quality and processes. By further exploring literature we have established direct links between the Manifesto for Agile Software Development and our theory for 2 out of 4 values and 5 out of 12 principles. Using a post hoc literature survey we have verified some of the established relations of the exploratory study.

Team Empowerment requires teams to be self-sufficient, have a corporate authority and take responsibility to enact their own decisions. Agile software development enhances the ability of a team to empower itself in multiple ways. It helps in maintaining a sustainable pace, enabling a team culture to empower itself. It creates awareness of non-functionals and allows the team to structure itself to maximise velocity. In turn, because developers of an empowered team have not been put under pressure unnecessarily, they are able to write software without rushing code. Moreover, empowered teams keep improving their skills and code which improves code quality. Furthermore, code quality adds value the user does not see. Raising awareness why you should have it helps in showing the importance of code quality and increases the chance that the opportunity is given to work on it. Finally, clear processes help the team in understanding what is expected of them and what they can expect. These agreements also apply to the code structure which increases the uniformity and comprehension of the code.

**RQ2:** How does the theory about code quality in Agile software development compare to global experience?

The answer to RQ2 is that we can generalise the results outside of the scope of the organisations we interviewed on a global scope. However, the results can only be generalised for Agile teams with a similar setup to the ones we studies. This means that the results hold for co-located teams that can change some of their processes themselves.

Using a survey we have corroborated or results from the Grounded Theory and made adjustments to make the theory applicable in a global setting. The respondents agreed to the questions with a total average median of 4.17. 4 questions had a median of 5.0, 1 a median of 4.5, 18 a median of 4.0 and 1 a median of 3.5. Data of the interviews and the survey were combined to increase the generalisability of the theory.

Responses were individually analysed to find answers that contradicted or indicated a lack in the theory. Contradictions that were given multiple times were compared against the data find out what caused these contradictions. One contradiction was found that caused a change in the theory. Moreover, answers that indicated a lack in the theory were compared with the interview data to verify if we missed phenomena or if the phenomena did not occur repeatedly in the interviews which made us discard it at the time of the coding.

**RQ3:** Which Agile software development practices impact code quality and how should they be implemented to enhance code quality?

We transformed the practices from the theory to best practices. Additional literature was reviewed to establish more best practices that fit in the theory. We determined the existence of nine best practices that can be used in Agile software development processes to enhance code quality which are as follows: Continuous Integration, Definition of Done, Direct Communication, Eliminate Technical Debt, Feedback, Involve All Parties in Development, Refactoring, Team Agreements, Version Control System. Section 7.1 complemented these nine best practices with a description how they can be implemented to enhance code quality.

**RQ4:** What are best practices in Agile software development methodologies, how are they linked to (sub)characteristics of code quality and how can they be used to increase quality scores through enhancing Agile processes?

Table 7.1 gives an answer to this RQ4. We combined each best practice with ISO/IEC 25010:2011 metrics in a table. The rows of the table consist of the nine best practices we concluded at RQ3. The columns of the table are the code quality characteristics of the ISO/IEC 25010:2011 standard which can be automatically measured i.e. Maintainability, Performance Efficiency, Security and Reliability. If a team has a low score for one of these four metrics, it can query the table which best practices it can use, or obtain insights into possible modifications for its the Agile software development processes.

# Chapter 9

# Discussion

In order to discuss our approach and the results, we describe external and internal threats. External validity defines which population the results can be generalised to. For example, our results hold for software development but not for building a house. Internal validity defines to what degree we were able to overcome challenges within the study itself. For example, a bias of the researcher. Limitations of the study will be discussed. Finally, we discuss how the research can be expanded in future work.

## 9.1 Threats to Validity

**Selection of participant's background might influence results.** A threat to external validity is the fact that we interviewed employees from only two companies. Participants of those companies could share a similar bias caused by the company culture. To counter this we took great care in selecting the participants of the companies. As already mentioned, one of the two organisations outsources some of their employees. We tried to approach employees that were outsourced to different clients to gain insights into that client's company culture. This was possible because outsourced employees join in the client's way of working which provides its perspective. We also made sure that all of our questions were unbiased. Moreover, the questionnaire ensured that the final theory was based on a group that is more varied.

**Not all people are equally articulate and perceptive.** The second threat to external validity is the fact that not all interviewees are equally articulate and perceptive. We tried minimising this by asking open-ended questions and giving the participants freedom in answering. If a question was not clear we rephrased it until the participant understood it. Furthermore, since not all participants were equally, articulate we tried interviewing multiple participants that had a similar role and similar experience.

**Researcher background influences results.** A threat to internal validity is the fact that the background of the researcher can generate a bias which might influence the results of the Grounded Theory. This was one of the reasons why we have not done an extensive literature survey and instead read only definitions i.e. Scrum guide and the ISO/IEC 25010:2011 standard.

Additionally, concepts were shared and discussed with multiple researchers at regular intervals to minimise bias and to reduce erroneous interpretations.

**Business interests influences results.** The second internal threat is the fact that this study was part of a Master's thesis and was written during an internship at an organisation. Because of the interest the organisation could take in the study, it may influence the results or the researcher. For Grounded Theory to succeed the researchers should not be influenced by anything other than the results. Interests of the organisation can conflict with that. Fortunately, the supervisor of the organisation gave us full freedom to perform our study. Moreover, any parties that could have influenced the results were neither contacted nor informed before our final presentation.

**Research lacks a structured literature review process.** The third internal threat is the fact that this thesis lacks a structured literature review process. Structured literature reviews are used to provide an exhaustive summary of current evidence relevant to the research questions. The fact that our initial research questions were used as a general guideline, a structured literature review could have been used to find all the available data. However, following the Grounded Theory we wanted to be unbiased before the research. Therefore, no structured literature review had to be done since a less exhaustive literature review was sufficient for this thesis.

## 9.2 Limitations of the Study

**Limitation of model applicability.** A limitation of Grounded Theory is that the resulting theory can only be applied to the specific context, as was explored in the study of Hoda et al. [34]. Because our field of study contained a high number of variables and a lack of examples of failure we could not define factors for failure in a project. However, we repeatedly observed that team empowerment was key in building code quality in Agile software development processes.

**Incompleteness of model** In the methodology we mentioned that GT lacks validation of theory. We enhanced the methodology by adding a verification phase. However, this provided validation for the interpretation and not for the results on which the theory was built. Moreover, this study is one of the first to explore the relation between code quality and Agile software development. Therefore, we cannot guarantee the completeness of our theory. Although we have discussed the validity of our interpretations of the data, the model is not complete. It is likely that other researchers that perform similar studies will have similarities and dissimilarities in their model compared to ours.

**Lack of testing the practical results.** We have defined best practices to increase code quality. However, we were not able to verify them in the long term. Ideally, we test each practice in existing projects and test if the metrics change as expected.

# Chapter 10

# Conclusion and Future Work

The goal of this study is to explore relationships between code quality and Agile software development and provide a solution to increase code quality in Agile software development. In this thesis, we have described our exploratory qualitative phase followed by a practical phase. In the qualitative phase, we used Grounded Theory in combination with 20 interviews. We concluded that Team Empowerment is the core relation between code quality and Agile software development. A post hoc analysis was done to verify the theory. The purpose of the practical phase was to describe best practices that can be used to enhance specific code quality characteristics of a project. We then revisited our four research questions and answered them in Chapter 8.

We presented samples of our data to demonstrate the outcomes of the Grounded Theory research. We described that the core category, team empowerment, contains 4 values; Sustainable Pace, Awareness, Team Culture and Team Structure. These are the main values an empowered team should possess. We further analysed these values and established conditions and practical implementations for achieving these values.

The theory has been converted to a questionnaire to verify if it is applicable outside the scope of the organisations that were interviewed. The results of the questionnaire show that the participants agree with the statements which were directly linked to parts of the theory and gave useful insights where the theory could be improved.

A final, practical phase combined all data and new literature to establish best practices. Developers can use these to modify their processes to enhance code quality through team empowerment. Similarly to the Scrum framework, teams can decide which practice works for them and implement it or get insights why their current processes are causing code quality related problems and which direction can be taken to fix them.

We are confident that the results of the Grounded Theory provide a solid foundation for other studies in the field of code quality in Agile software development. Although we cannot guarantee the completeness of our results it will both provide insights into concrete values and conditions that can be further studied, and offer a basis on which further research can be built.

## 10.1 Future Work

**Similar Grounded Theory research to complement results.** This thesis performed a Grounded Theory study. However, Glaser [31] states that the focus of GT is the generalisation of theory and that validation may be undertaken by different researchers using other methods. Although we provided additional validation via the quantitative phase, we are aware of the limitations of our study in the sense that the model is incomplete. Future work can be done by other researchers who conduct their own Grounded Theory study after which the results can be combined with those of this thesis. Additionally, the size of the development team and the role of the Product Owner are phenomena related to code quality that can be researched in future studies.

**Code quality in Globally Distributed Agile Software Engineering.** The conducted study focused on teams who were either co-located or working close to each other, allowing them to have regular face-to-face meetings. Since a significant factor in our results is the communication, similar research can be done for teams that are separated by large distances e.g. different continents.

**Similar Grounded Theory in other Agile software development methodologies/frameworks.** This research was a response to the lack of literature about the impact on code quality when within the Agile software development process. Consequently, the results of our study provide a global overview of this. Further research is required in the values to further explore their conditions and define best practices in the Agile way of working to enhance the code quality. Another perspective that can be added to future work is that of different Agile software development methodologies since our interviewees mainly worked with the Scrum framework.

**Test results in existing projects.** We discussed that a limitation of our study is the lack of practical testing of the best practices. Future work can implement one or more of these practices and follow how the metrics of a project change over time. If the links are correct we should see improvements. Moreover, future research can also study how best practices influence each other or what other effects of best practices are since it is likely that not only the code quality will change but also the velocity of the team for example.

# Bibliography


[1] Mark Aberdour. Achieving quality in open-source software. *IEEE software*, 24(1), 2007.

[2] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*, 2017.

[3] Steve Adolph, Wendy Hall, and Philippe Kruchten. Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16(4):487–513, 2011.

[4] A Ahmed, S Ahmad, N Ehsan, E Mirza, and SZ Sarwar. Agile software development: Impact on productivity and quality. In *Management of innovation and technology (ICMIT), 2010 IEEE international conference on*, pages 287–291. IEEE, 2010.

[5] I Elaine Allen and Christopher A Seaman. Likert scales and data analyses. *Quality progress*, 40(7):64–65, 2007.

[6] Scott Ambler. Quality in an agile world. *Software Quality Professional*, 7(4):34, 2005.

[7] Robert D Austin. The effects of time pressure on quality in software development: An agency model. *Information systems research*, 12(2):195–207, 2001.

[8] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. *[online] Agilemanifesto.org*, 2001.

[9] Andrew Begel and Nachiappan Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pages 255–264. IEEE, 2007.

[10] Sonali Bhasin. Quality assurance in agile: a study towards achieving excellence. In *AGILE India (AGILE INDIA), 2012*, pages 64–67. IEEE, 2012.

[11] Barry W Boehm, John R Brown, and Mlity Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605. IEEE Computer Society Press, 1976.

[12] Norman M Bradburn, Seymour Sudman, and Brian Wansink. *Asking questions: the definitive guide to questionnaire design–for market research, political polls, and social and health questionnaires*. John Wiley & Sons, 2004.

[13] Lionel C Briand, Jürgen Wüst, John W Daly, and D Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of systems and software*, 51(3):245–273, 2000.

[14] Antony Bryant and Kathy Charmaz. *The Sage handbook of grounded theory*. Sage, 2007.

[15] Erran Carmel and Ritu Agarwal. Tactical approaches for alleviating distance in global software development. *IEEE software*, 18(2):22–29, 2001.

[16] Kathy Charmaz. *Constructing grounded theory*. Sage, 2014.

[17] Alistair Cockburn. *Agile software development: the cooperative game*. Pearson Education, 2006.

[18] Jason Cohen, Eric Brown, Brandon DuRette, and Steven Teleki. *Best kept secrets of peer code review*. Smart Bear Somerville, 2006.

[19] Juliet Corbin, Anselm Strauss, and Anselm L Strauss. *Basics of qualitative research*. Sage, 2014.

[20] Juliet M Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology*, 13(1):3–21, 1990.

[21] John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[22] Noopur Davis. Driving quality improvement and reducing technical debt with the definition of done. In *Agile Conference (AGILE), 2013*, pages 164–168. IEEE, 2013.

[23] William Edwards Deming and Deming W Edwards. *Quality, productivity, and competitive position*, volume 183. Massachusetts Institute of Technology, Center for advanced engineering study Cambridge, MA, 1982.

[24] Steve Denning. Agile: The world's most popular innovation engine. *Forbes*, 2015.

[25] Karthik Dinakar. Agile development: overcoming a short-term focus in implementing best practices. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 579–588. ACM, 2009.

[26] Oliver Dolan. How do agile teams ensure code quality? `https://www.quora.com/How-do-agile-teams-ensure-code-quality`, 2017.

[27] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[28] Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50(9-10):833–859, 2008.

[29] Patrick M Erwin. Corporate codes of conduct: The effects of code content and quality on ethical performance. *Journal of Business Ethics*, 99(4):535–548, 2010.

[30] Barney Glaser. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

[31] Barney G Glaser. *Basics of grounded theory analysis: Emergence vs forcing*. Sociology press, 1992.

[32] Barney G Glaser. *Doing grounded theory: Issues and discussions*. Sociology Press, 1998.

[33] James Herbsleb, David Zubrow, Dennis Goldenson, Will Hayes, and Mark Paulk. Software quality and the capability maturity model. *Communications of the ACM*, 40(6):30–40, 1997.

[34] Rashina Hoda, James Noble, and Stuart Marshall. Developing a grounded theory to explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17(6):609–639, 2012.

[35] Ming Huo, June Verner, Liming Zhu, and Muhammad Ali Babar. Software quality and agile methods. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 520–525. IEEE, 2004.

[36] Azham Hussain and Emmanuel OC Mkpojiogu. An application of iso/iec 25010 standard in the quality-in-use assessment of an online health awareness system. *Jurnal Teknologi*, 77(5):9–13, 2015.

[37] Capers Jones. *Software engineering best practices*. McGraw-Hill, Inc., 2009.

[38] Maarit Laanti, Outi Salo, and Pekka Abrahamsson. Agile methods rapidly replacing traditional methods at nokia: A survey of opinions on agile transformation. *Information and Software Technology*, 53(3):276–290, 2011.

[39] Jean-Louis Letouzey. The sqale method for evaluating technical debt. In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 31–36. IEEE, 2012.

[40] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kahkonen. Agile software development in large organizations. *Computer*, 37(12):26–34, 2004.

[41] Peter Meso and Radhika Jain. Agile software development: adaptive systems principles and best practices. *Information systems management*, 23(3):19–30, 2006.

[42] Audris Mockus, Roy T Fielding, and James D Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, 2002.

[43] Nils Brede Moe, Torgeir Dingsøyr, and Tore Dybå. Understanding self-organizing teams in agile software development. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 76–85. IEEE, 2008.

[44] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The influence of organizational structure on software quality. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 521–530. IEEE, 2008.

[45] Sofia Ouhbi, Ali Idri, José Luis Fernández Alemán, Ambrosio Toval, and Halima Benjelloun. Applying iso/iec 25010 on mobile personal health records. In *HEALTHINF*, pages 405–412, 2015.

[46] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008.

[47] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[48] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 155–165. ACM, 2014.

[49] Patrick J Roache. *Verification and validation in computational science and engineering*, volume 895. Hermosa Albuquerque, NM, 1998.

[50] Álvaro Rocha. Framework for a global quality evaluation of a website. *Online Information Review*, 36(3):374–382, 2012.

[51] Ioannis Samoladas, Ioannis Stamelos, Lefteris Angelis, and Apostolos Oikonomou. Open source software development should strive for even greater code maintainability. *Communications of the ACM*, 47(10):83–87, 2004.

[52] Norman F. Schneidewind. Methodology for validating software metrics. *IEEE Transactions on software engineering*, 18(5):410–422, 1992.

[53] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[54] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[55] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 21, 2011.

[56] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Guides*, 2017.

[57] Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12 (1):43–60, 2002.

[58] Christoph J Stettina and Werner Heijstek. Five agile factors: Helping self-management to self-reflect. In *European Conference on Software Process Improvement*, pages 84–96. Springer, 2011.

[59] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. Grounded theory in software engineering research: a critical review and guidelines. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 120–131. IEEE, 2016.

[60] Anselm Strauss and Juliet Corbin. Grounded theory methodology. *Handbook of qualitative research*, 17:273–285, 1994.

[61] Anselm Strauss and Juliet M Corbin. *Basics of qualitative research: Grounded theory procedures and techniques.* Sage Publications, Inc, 1990.

[62] Anselm L Strauss. *Qualitative analysis for social scientists*. Cambridge University Press, 1987.

[63] Diane E Strode, Sid L Huff, and Alexei Tretiakov. The impact of organizational culture on agile method use. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–9. IEEE, 2009.

[64] Arie Van Deursen and Tobias Kuipers. Source-based software risk assessment. In *null*, page 385. IEEE, 2003.

[65] Richard Vidgen and Xiaofeng Wang. Coevolving systems and the organization of agile software development. *Information Systems Research*, 20(3):355–376, 2009.

[66] Xinyu Wang, Liping Zhao, Ye Wang, and Jie Sun. The role of requirements engineering practices in agile development: an empirical study. In *Requirements Engineering*, pages 195–209. Springer, 2014.

[67] Greg Wilson, Dhavide A Aruliah, C Titus Brown, Neil P Chue Hong, Matt Davis, Richard T Guy, Steven HD Haddock, Kathryn D Huff, Ian M Mitchell, Mark D Plumbley, et al. Best practices for scientific computing. *PLoS biology*, 12(1):e1001745, 2014.

[68] Yahaya Y Yusuf, Mansoor Sarhadi, and Angappa Gunasekaran. Agile manufacturing:: The drivers, concepts and attributes. *International Journal of production economics*, 62(1-2): 33–43, 1999.

# Appendix A

# Consent to Participate Form

**Consent to Participate in a Research Study**
**Delft University of Technology • Delft, NL**

**Title of Study:** **Exploring the Relation Between Agile Software Development and Code Quality**

**Researcher:**

**Name:** **Lars Krombeen** **Dept:** SE **Phone:** [redacted]

**Introduction**
- You are being asked to be in a research study of code quality in agile software development.
- We ask that you read this form and ask any questions that you may have before agreeing to be in the study.

**Purpose of Study**
- The purpose of the study is to define a theory about Code Quality in Software Development. Using constant comparison, theories will emerge which are tested against new data obtained by interviewing people.
- Ultimately, this research is presented as a paper and is part of a MSc graduation thesis.

**Description of the Study Procedures**
- If you agree to be in this study, you will be asked to participate in an interview lasting approximately 60 minutes in which factors for code quality are discussed.

**Risks/Discomforts of Being in this Study**
- There are no risks/discomforts of being is this study.

**Benefits of Being in the Study**
- By participating to this study, you help giving practical insights in understanding the impact of software development factors on code quality, and help an MSc student in doing his thesis project.

**Confidentiality**
- The records of this study will be made anonymous and kept strictly confidential. Audio and possibly video will be recorded during the interview and immediately modified after to only contain audio. Video and other identifiable data is removed. Research records will be kept encrypted and secured using a password protected file. Your audio record will be used for educational purposed only and is accessible by the undersigned. We will not include any information in any stored data or reports we may publish that would make it possible to

identify you. Your audio recordings are not kept for longer than 6 months and are destroyed before then.

**Right to Refuse or Withdraw**

- The decision to participate in this study is entirely up to you. You may refuse to take part in the study *at any time*. You have the right not to answer any single question, as well as to withdraw completely from the interview at any point during the process; additionally, you have the right to request that the interviewer not use/delete any of your interview material *at any time*.

**Right to Ask Questions and Report Concerns**

- You have the right to ask questions about this research study and to have those questions answered by me before, during or after the research. If you have any further questions about the study, at any time feel free to contact me, Lars Krombeen at l.krombeen@student.tudelft.nl or by telephone at [redacted].

**Consent**

- Your signature below indicates that you have decided to volunteer as a research participant for this study, and that you have read and understood the information provided above. You will be given a signed and dated copy of this form to keep.

Subject's Name (print): _____

Subject's Signature: _____ Date: _____

Investigator's Signature: _____ Date: _____

# Appendix B

# Example of Axial Coding Components

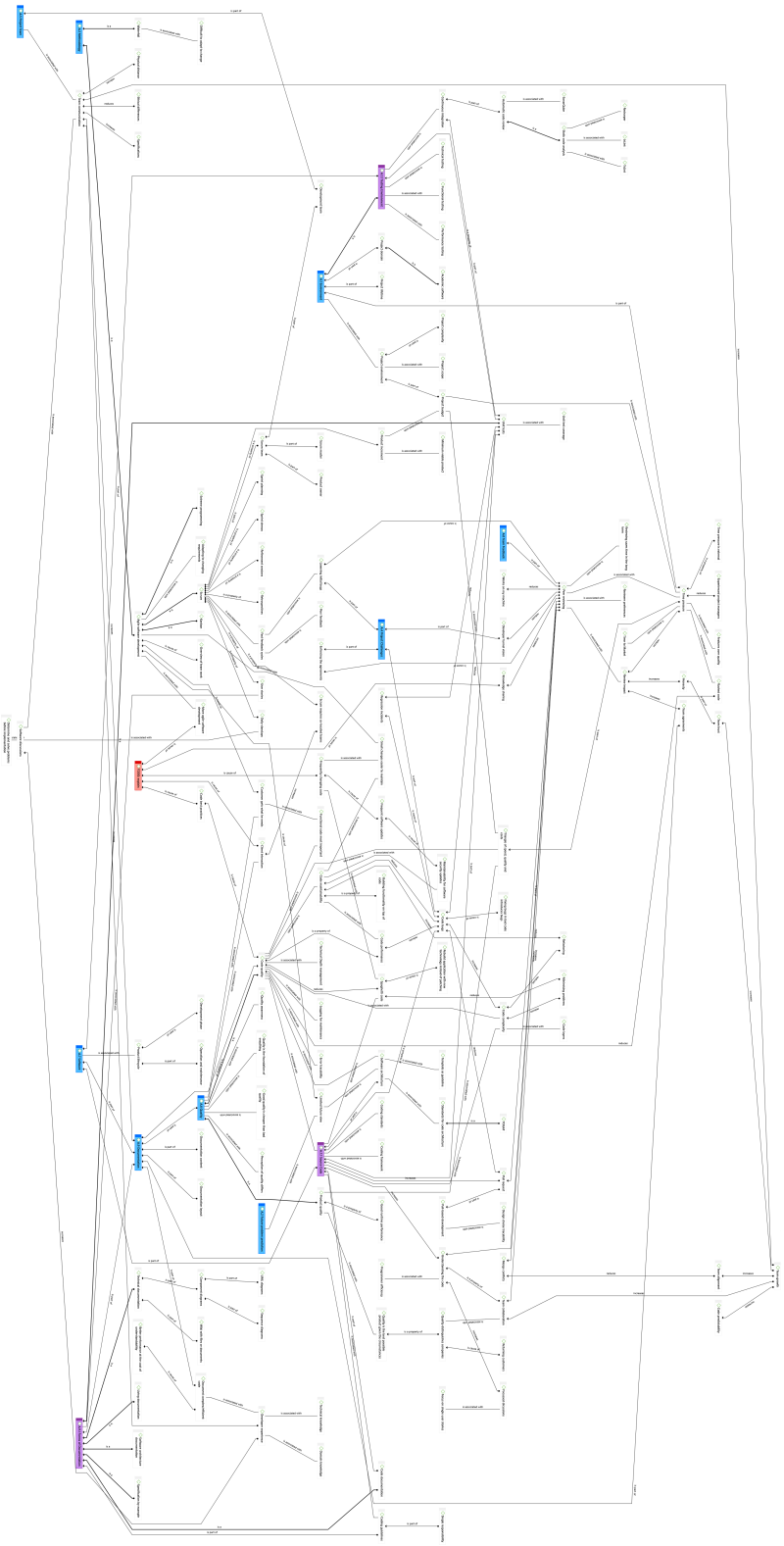

Figure B.1: Example of our Axial Codes and Components after the first two interviews.

# Appendix C

# Concept and Category Definitions

Table C.1: Grounded Theory Concepts and Categories, and their definitions.

| ID | Concept / Category | Definition |
|---|---|---|
| 1 | Agile awareness | Agile is an iterative way of working where the customer gets what he needs which is caused by interactive development. Involved parties should be made aware of how this way of working is executed and why. This starts at the sales department and ends with the operation and maintenance cycle. |
| 2 | Balance between focus & communication | Open communication in the team helps in solving problems but potentially reduces the focus of developers. There is a balance where helpful communication is becoming too interrupting. |
| 3 | Balance between priorities and necessities | There is a balance between working on features that have to be built and working on what the product actually needs. |
| 4 | Challenge yourself | Keep pushing your limits to keep the work interesting and new, but know your limits. |
| 5 | Code quality saves time | Code with a high quality saves time and budget in the long term. |
| 6 | Commit to be responsible | Take responsibility for your work and for the product. As a team you commit to the goal to deliver as promised. |

| | | |
|---|---|---|
| 7 | Continuous feedback | The interaction between the development team and the stakeholders allows the team to respond to changing user needs as they emerge to build what the customer needs. |
| 8 | Culture | Team culture determines how people think and interact with each other in a team. |
| 9 | Feeling comfortable in the team | Developers should feel safe in the team and with the tasks they are working on. |
| 11 | Flexibility | A structure does not violate flexibility. As the team grows the events can take be shortened or mutate altogether. |
| 12 | Involving all parties in development | All parties that are involved in the process of the development of the applications should be aware why agile is being done, what this means for them and how they will benefit from it when applied correctly. They are involved in the development of the product. |
| 13 | Iteration pressure | Iterations can give pressure because, at the end of the week developers want to finish the goal they are committed to. |
| 14 | Keep learning and improving code | Do not fall behind on quality or make sure that if you do, you catch up. Additionally, keep improving yourself as a developer. |
| 15 | Maintainability | Using the data, maintainability means that code is easy to extend or modify without breaking the program or taking too much time to understand what is happening. |
| 16 | Minimise pressure | Deadlines are the enemies of code quality. Deadlines cause code to be rushed/hacked. These temporary fixes are often forgotten, reducing the overall quality of the code. |
| 17 | Non-functional awareness | All parties should be aware of what it means to have quality (not only code quality). This starts at the beginning of the project and ends with the delivery. The customer knows that the requirements will change, that not everything can be known beforehand and the customer should adapt to change just like the development team does. |
| 18 | Open communication | Individuals are open and honest with each other and with the stakeholders to avoid unnecessary surprises. |

| | | |
|---|---|---|
| 19 | Peer Feedback | Peer feedback is a method developers can use to learn from each other and to improve each other when they see something that is not built as they agreed upon or could have been built better. |
| 20 | Programmer efficiency | Efficiency is the velocity at which programmers can work on the code. An efficient programmer takes less time to implement a ticket with the same quality as a developer that is not efficient. |
| 21 | Self-organising teams | Teams can organise themselves and have the authority to take decisions to work towards the common goal. |
| 22 | Splitting complex tasks from sprint | Not all the work can be done in iterations. Complex documents like the architecture should not be done as tasks in a sprint. The sprints can be used as a way to add feedback to these documents but they are not tasks in the sprint. |
| 23 | TD Management | Technical debt management is an ordered document. The team should be given time to work on technical debt. Ideally no technical debt is created. However, a release date might force a developer to rush/hack some code. At this point the developer makes a new entry describing the design choice and the reason behind it in the TD document. |
| 24 | Team agreements | Team agreements are agreements such as which coding guidelines / best practices are followed, what framework/methodology they follow, how/what to document (e.g. design choices, architecture, etc.) and how to conduct the requirements analysis and how to work with or adjust it. |
| 25 | Team empowerment | As a team you are responsible for the product. It has the authority to make decisions and consists of the required people to finish their goals. |
| 26 | Team openness | The team is open to each other and to the stakeholders. If they think they cannot meet expectations they tell it honestly. If there is an impediment they let the right people know. Being open builds trust between parties. |
| 27 | Team predictability | You know what your team is doing and what you can expect from them and vice versa. Knowing what to expect increases team efficiency. |

| | | |
|---|---|---|
| 28 | Team structure | Agile enhances the communication in the development team. Having iterations allows teams to have repetitive events at which they can discuss everything they need to maximise their efficiency. There are agile frameworks that give a structure to a team and project to achieve that. The agreements that were made in the agile way of 2 working have to be kept. |
| 29 | Understanding the code | When scrolling through the code a developer that has not previously worked on that part should understand what is happening without spending too much time on it. |
| 30 | Why agile $>$ doing agile | It is more important to understand why you are doing agile than strictly following the rules. |
| 31 | Work floor discussions | Discussions are face to face and team members can easily ask each other for help. Work floor discussions increase the speed at which answers are given and reduce misinterpretations. |

# Appendix D

# Interview Questions

This appendix will quote most questions the researcher asked during the interviews. Questions that were used as introduction e.g. how long have you been employee at company X, are not given because they are non-relevant. All questions were originally in Dutch and were translated to English.

Table D.1: Key Questions asked by the interviewer

| **ID** | **Participant** | **Question Quotation** |
|---|---|---|
| Q1 | P1 | *"What is your definition of code quality? "* |
| Q2 | P1 | *"The points that you mentioned, why would you do that? What does the team gain with it? "* |
| Q3 | P1 | *"Have you ever seen it go wrong? "* |
| Q4 | P1 | *"Did you work in a team or with just the two of you? "* |
| Q5 | P1 | *"In the case you mentioned, would it have gone differently when you would have worked in a Waterfall approach instead of an Agile one? "* |
| Q6 | P1 | *"Would it be a problem for the software? "* |
| Q7 | P1 | *"What do you think about that? "* |
| Q8 | P1 | *"To what degree do you think that maintainability is important? "* |
| Q9 | P1 | *"Do you think that there are other factors in Scrum that impact your code quality? Maybe you can give some examples."* |
| Q10 | P1 | *"So mainly the environment and how much you are pushed? "* |
| Q11 | P1 | *"If you do not have those factors, how would you organise your process to obtain a high code quality? "* |
| Q12 | P1 | *"Can you think of any other factors? "* |
| Q13 | P1 | *"What do you mean with reviewing? "* |
| Q14 | P1 | *"Is there a gradation to what degree you can review? "* |

| | | |
|---|---|---|
| Q15 | P1 | *"What should you consider when wanting to analyse and understand everything? "* |
| Q16 | P1 | *"Do you feel that Scrum changes how the team collaborates? "* |
| Q17 | P1 | *"Do you think Scrum helps with that? "* |
| Q18 | P1 | *"Could you give an example of that? "* |
| Q19 | P1 | *"How would you arrange the team agreements and how can Scrum help with that? "* |
| Q20 | P1 | *"How can you enforce the agreements? "* |
| Q21 | P1 | *"Did you work on a project without reviewing and on a project with reviewing? Did you see a difference in code quality? "* |
| Q22 | P1 | *"So when you are not doing reviews you also have to spend more time on, for example merge conflicts, or double work? "* |
| Q23 | P1 | *"Did that group use the Scrum framework? "* |
| Q24 | P1 | *"Give this graph with time versus groundedness of reviewing, could you draw a line? "* |
| Q25 | P1 | *"What would you recommend, no reviews or grounded reviews? "* |
| Q26 | P1 | *"Can you draw another graph for efficiency versus maintainability of the code? "* |
| Q27 | P1 | *"Do you think that Scrum can help in improving the maintainability? "* |
| Q28 | P1 | *"And what if you use Scrum? "* |
| Q29 | P2 | *"What is your definition of code quality? "* |
| Q30 | P2 | *"Why would you deliver quality? "* |
| Q31 | P2 | *"Could you tell me more about that? "* |
| Q32 | P2 | *"Do you think that there are differences in quality for the customer and the internal teams? "* |
| Q33 | P2 | *"How do you make sure quality is implemented in the entire pipeline? "* |
| Q34 | P2 | *"How do you feel about working with it? "* |
| Q35 | P2 | *"What do you like or dislike about your work environment? "* |
| Q36 | P2 | *"How important is the documentation for the final product? "* |
| Q37 | P2 | *"Do you think that the culture of a country impacts the quality? "* |
| Q38 | P2 | *"How do you feel about Agile? "* |
| Q39 | P2 | *"Do you like working with Agile? "* |

| | | |
|---|---|---|
| Q40 | P2 | *"Do you think that there are Agile factors that impact quality? "* |
| Q41 | P3 | *"What is your definition of code quality? "* |
| Q42 | P3 | *"Let us continue on that. Understanding the code and efficient code. Do you think those values can collide? "* |
| Q43 | P3 | *"Do you think that they are related? "* |
| Q44 | P3 | *"How do you experience quality in your work? "* |
| Q45 | P3 | *"You said that quality in [] is not that good. How do you see that? "* |
| Q46 | P3 | *"Given an unlimited amount of time and budget, how would you arrange your processes to achieve a high code quality? "* |
| Q47 | P3 | *"Could you think of something else? "* |
| Q48 | P3 | *"So you say that team building is important? "* |
| Q49 | P3 | *"How do you stimulate such an environment? "* |
| Q50 | P3 | *"To what extend does education impact code quality? "* |
| Q51 | P3 | *"How does that lead to the code quality you want to achieve? "* |
| Q52 | P3 | *"So that is also the aspect that you do not look far ahead enough? "* |
| Q53 | P3 | *"How would you stimulate that? "* |
| Q54 | P3 | *"What do you mean with growth? "* |
| Q55 | P3 | *"Given an unlimited amount of time and budget, how would you educate your team to achieve a high code quality? "* |
| Q56 | P3 | *"How do you feel about the Waterfall model? "* |
| Q57 | P3 | *"Could you mention some best practices? "* |
| Q58 | P3 | *"What do you achieve with it? "* |
| Q59 | P3 | *"So quality costs time but also saves time? "* |
| Q60 | P4 | *"How would you define code quality? "* |
| Q61 | P4 | *"What is readability? "* |
| Q62 | P4 | *"Do you think that code quality is important? "* |
| Q63 | P4 | *"Can you give some of those effects? "* |
| Q64 | P4 | *"Do you think that performance impacts code quality? So when you have to make a piece of code run faster."* |
| Q65 | P4 | *"How do you see code quality in your code? How do you make sure your code is of high quality? "* |
| Q66 | P4 | *"Can you think of other external factors that impact the quality of the code? "* |

| | | |
|---|---|---|
| Q67 | P4 | *"Are there positive factors on your code quality? "* |
| Q68 | P4 | *"Are there factors that you recommend, or want to change when possible? "* |
| Q69 | P4 | *"Can you think of other things to improve your code quality? "* |
| Q70 | P4 | *"How much time do you spend on reviewing? "* |
| Q71 | P4 | *"Are peer reviews useful for anything other than enhancing code quality? "* |
| Q72 | P4 | *"How do you feel about that? "* |
| Q73 | P4 | *"Could you give some of the disadvantages when you skip reviews? Or why you should not bother with them? "* |
| Q74 | P4 | *"What are considerations you can take for reviewing? "* |
| Q75 | P4 | *"Do you think you should do code reviews in teams with 5 members? "* |
| Q76 | P4 | *"How much time would it save on the project duration when you do reviews versus when you skip them? "* |
| Q77 | P4 | *"Do you think that it costs an equal amount of time? So no reviews or basic reviews? "* |
| Q78 | P4 | *"Do you think that Agile is one of the factors that impact code quality? "* |
| Q79 | P4 | *"How does that influence your code quality? "* |
| Q80 | P4 | *"What are considerations you take when you decide on doing a project with the Waterfall model or Agile? "* |
| Q81 | P4 | *"What are important forms of documentations to warrant code quality? "* |
| Q82 | P4 | *"How do you communicate team agreements? "* |
| Q83 | P5 | *"What is your definition of code quality? "* |
| Q84 | P5 | *"Could you give examples of that context? "* |
| Q85 | P5 | *"What does quality add for different parties? "* |
| Q86 | P5 | *"Where do you think that comes from? "* |
| Q87 | P5 | *"Where does that difference come from? "* |
| Q88 | P5 | *"What do you mean with that culture? "* |
| Q89 | P5 | *"Given an unlimited amount of time and budget, how would you stimulate the culture to enhance openness? "* |
| Q90 | P5 | *"What are advantages of the precautions you take? "* |
| Q91 | P5 | *"How would you manage that? "* |
| Q92 | P5 | *"Is it easy to include new members in the culture of the team? "* |

| | | |
|---|---|---|
| Q93 | P5 | *“So it is not really something individual? “* |
| Q94 | P5 | *“So new members are not intimidated by the team? “* |
| Q95 | P5 | *“Do you think that there is a difference between juniors and seniors? “* |
| Q96 | P5 | *“Did it ever you wrong? “* |
| Q97 | P5 | *“How can you tell? “* |
| Q98 | P5 | *“Given the scenario that you can lead a new team. How would you arrange your processes to get the team where you want it do be? What are the steps you would take? “* |
| Q99 | P5 | *“Can you think of a methodology that helps in doing that? “* |
| Q100 | P5 | *“Do you think that working from minimal requirements limits the ability to look ahead? “* |
| Q101 | P5 | *“Do you think that knowledge sharing influences the code quality the team delivers? “* |
| Q102 | P6 | *“What does code quality mean to you? “* |
| Q103 | P6 | *“Are there other factors that influence the efficiency of developers? “* |
| Q104 | P6 | *“Why is code quality important when looking at developer efficiency? “* |
| Q105 | P6 | *“Readability of the code, how can you see that in your code? “* |
| Q106 | P6 | *“Why does code have to be rewritten? “* |
| Q107 | P6 | *“Do you think that Scrum influences the fact that code has to be rewritten? “* |
| Q108 | P6 | *“Could you give examples of those external factors? “* |
| Q109 | P6 | *“Do you like that you have minimal contact with external parties and only collaborate with the product owner? “* |
| Q110 | P6 | *“Would you like to include the end-users in the development? “* |
| Q111 | P6 | *“Do you have other examples of that? “* |
| Q112 | P6 | *“What is your opinion about coding practices and guidelines? “* |
| Q113 | P6 | *“Which guidelines should you or should you not follow? “* |
| Q114 | P6 | *“Can you tell me something more about coding practices? “* |
| Q115 | P6 | *“How do you enforce those agreements? “* |
| Q116 | P6 | *“How do you make sure that bad code does not reach production? “* |
| Q117 | P6 | *“What is bad quality? “* |

| | | |
|---|---|---|
| Q118 | P6 | *"Are there other things that can happen by accident, or sneak in slowly? "* |
| Q119 | P6 | *"Are there other factors that cause bugs or unreadable code? "* |
| Q120 | P6 | *"How do you get to that level of communication? "* |
| Q121 | P6 | *"Those cubicles and managing space, is that an external or internal factor? "* |
| Q122 | P6 | *"Can you describe how pressure influences the communication of a team? "* |
| Q123 | P7 | *"What is your definition of code quality? "* |
| Q124 | P7 | *"Could you give examples of factors that increase or decrease the code quality? "* |
| Q125 | P7 | *"What do you mean with those procedures? "* |
| Q126 | P7 | *"How would you decide if a project is done using the Waterfall model or Agile? "* |
| Q127 | P7 | *"How do you get proper communication in teams that are not co-located? "* |
| Q128 | P7 | *"Are there other things you see that go wrong in communicating? "* |
| Q129 | P7 | *"Do you think that quality saves time in the long term? "* |
| Q130 | P7 | *"What are the most important factors in achieving that attitude? "* |
| Q131 | P8 | *"Can you give a brief summary of Agile and what it means for you? "* |
| Q132 | P8 | *"Do you like working with it? "* |
| Q133 | P8 | *"Why do you like the iterative approach? "* |
| Q134 | P8 | *"Can you think of other advantages of Agile? "* |
| Q135 | P8 | *"Can you think of other risks? "* |
| Q136 | P8 | *"How would you arrange processes to reduce that pressure? "* |
| Q137 | P8 | *"Is that good or bad? "* |
| Q138 | P8 | *"If you were responsible for the Agile processes of a new team, how would you arrange it? "* |
| Q139 | P8 | *"How would you deal with situations where you have to delay something? "* |
| Q140 | P8 | *"Did you ever see it go wrong, even with that margin? "* |
| Q141 | P8 | *"Do you think that there are Agile factors that impact the efficiency of developers? "* |
| Q142 | P8 | *"What would your ideal workspace be? "* |

| | | |
|---|---|---|
| Q143 | P8 | *“All by yourself? “* |
| Q144 | P8 | *“And what causes that? “* |
| Q145 | P8 | *“Does that also work for trivial pieces of code? “* |
| Q146 | P8 | *“Would you make any exceptions? “* |
| Q147 | P9 | *“What is your definition of code quality? “* |
| Q148 | P9 | *“Why would you implement code quality? “* |
| Q149 | P9 | *“What happens when you do not do it? “* |
| Q150 | P9 | *“What are reasons that cause a developer to rush code? “* |
| Q151 | P9 | *“So technical debt? “* |
| Q152 | P9 | *“What can you do to make sure that technical debt is solved? “* |
| Q153 | P9 | *“Can you think of other factors that cause developers to write bad code? “* |
| Q154 | P9 | *“What would be a good mentality to reduce external pressure? “* |
| Q155 | P9 | *“How do you feel about Agile? “* |
| Q156 | P9 | *“So you dislike Agile because it enforces a way of working? “* |
| Q157 | P9 | *“Do you like that you are able to respond quickly to change? “* |
| Q158 | P10 | *“Can you give a brief summary of Agile and what it means for you? “* |
| Q159 | P10 | *“Do you like working with it? Why? “* |
| Q160 | P10 | *“Can you think of other advantages, or maybe disadvantages? “* |
| Q161 | P10 | *“If you were able to change it, how would you do it instead? “* |
| Q162 | P10 | *“Are the Agile factors that have a positive or negative effect on the efficiency of developers? “* |
| Q163 | P10 | *“Are there processes of the Scrum framework that can be included in these factors? “* |
| Q164 | P10 | *“How would you document the code to make sure that it still can be understood after 2 years? “* |
| Q165 | P10 | *“Do you also coach in the Agile processes? “* |
| Q166 | P10 | *“Are there things that frequently go wrong? “* |
| Q167 | P11 | *“What would be an ideal state of an application you receive to maintain? “* |
| Q168 | P11 | *“With documentation, do you mean a guide that explains how everything works? “* |
| Q169 | P11 | *“What do you mean with clear? “* |

| | | |
|---|---|---|
| Q170 | P11 | *"Can you give a brief summary of Agile and what it means for you? "* |
| Q171 | P11 | *"How do you feel about that? "* |
| Q172 | P11 | *"You mentioned more efficient, what causes that? "* |
| Q173 | P11 | *"Are there other events in the Scrum framework? "* |
| Q174 | P11 | *"Do these events influence your efficiency? "* |
| Q175 | P11 | *"Can you give an example where you changed some of the Agile processes because you are working in the same team for a long time? "* |
| Q176 | P11 | *"Can you think of advantages or disadvantages of Agile? "* |
| Q177 | P11 | *"How can you convince a client to join in the Scrum way of working? "* |
| Q178 | P11 | *"How would you implement Scrum if you had to design the Agile processes? "* |
| Q179 | P11 | *"You mentioned that you have to be more strict towards your client, how would you tackle that? "* |
| Q180 | P13 | *"Can you give a brief summary of Agile and what it means for you? "* |
| Q181 | P13 | *"Do you like working with Scrum? "* |
| Q182 | P13 | *"If you are flexible, do you still adhere to some kind of structure? "* |
| Q183 | P13 | *"Could you think of other advantages of Scrum? "* |
| Q184 | P13 | *"Can you think of disadvantages? "* |
| Q185 | P13 | *"When you are coaching in Scrum, are there aspects you focus on more than others? "* |
| Q186 | P13 | *"Do you like that the team is responsible? "* |
| Q187 | P13 | *"How do you feel about the self-organising principle? "* |
| Q188 | P13 | *"Do you think that is is better than when someone is managing it from upper management? "* |
| Q189 | P13 | *"Can you think of other advantages? "* |
| Q190 | P13 | *"Can you think of disadvantages? "* |
| Q191 | P13 | *"What is more important when coaching Agile, making sure that the team understands Scrum or understands and follows the processes? "* |
| Q192 | P13 | *"What would happen is someone understands what Scrum is but does now know why you are using it? "* |

| | | |
|---|---|---|
| Q193 | P14 | *“Could you give examples of Agile factors that improve your ability to write good code? “* |
| Q194 | P14 | *“Are there factors that you do not like? “* |
| Q195 | P14 | *“Would you like to change anything to further improve your efficiency? “* |
| Q196 | P14 | *“Why do you think that? “* |
| Q197 | P14 | *“What do you mean with deadline? Can you be more specific? “* |
| Q198 | P14 | *“What is your knowledge of Scrum? “* |
| Q199 | P14 | *“How do you feel about Scrum? “* |
| Q200 | P14 | *“Can you give an example when it went wrong? “* |
| Q201 | P14 | *“How would you have prevented it? “* |
| Q202 | P14 | *“Can you think of other Scrum processes that improve your code quality? “* |
| Q203 | P14 | *“Can you think of disadvantages? “* |
| Q204 | P14 | *“You mentioned that you like the freedom that you have to solve problems. Do you see something of that in the Scrum way of working? “* |
| Q205 | P14 | *“Do you think that there are factors in Scrum that increase your efficiency? “* |
| Q206 | P14 | *“What would you change in Scrum to keep your efficiency as high as possible? “* |
| Q207 | P14 | *“So Waterfall is rushing? “* |
| Q208 | P14 | *“Do you think that it is caused by deadlines? “* |
| Q209 | P15 | *“Can you give a brief summary of Agile and what it means for you? “* |
| Q210 | P15 | *“You mentioned that Scrum increases the efficiency. Can you elaborate? “* |
| Q211 | P15 | *“How would you change Scrum to increase your efficiency? “* |
| Q212 | P15 | *“Did you ever see things go wrong in a Scrum team? “* |
| Q213 | P15 | *“Do you think that Scrum impacts to quality of the work you do, or how fast you can build? “* |
| Q214 | P15 | *“Can you think of disadvantages? “* |
| Q215 | P15 | *“What do you think of the responsibility principles of Scrum? “* |
| Q216 | P15 | *“Can you see that in your code quality? “* |
| Q217 | P15 | *“With functionality, you mean what the customer wants? “* |

| | | |
|---|---|---|
| Q218 | P15 | *"What would happen if you do not get the time to work in that? "* |
| Q219 | P15 | *"Does that make the customer happy? "* |
| Q220 | P15 | *"Why is it a pity when you are not given the time to work on it? "* |
| Q221 | P15 | *"Which role does the development team have to raise that awareness? "* |
| Q222 | P15 | *"When something goes wrong in the team, do you have the ability to solve it internally? "* |
| Q223 | P15 | *"Would you like to have contact with higher management when you look at it from a perspective of code quality? "* |
| Q224 | P15 | *"It costs a lot of time, but why do you say that is increases the efficiency in the end? "* |
| Q225 | P15 | *"It is more important to follow structure, or be flexible with it and adapt to the team needs? "* |
| Q226 | P15 | *"Why do people need training in how to use Agile before it impacts the efficiency? "* |
| Q227 | P16 | *"Can you give a brief summary of Agile and what it means for you? "* |
| Q228 | P16 | *"Why is it not cheaper? "* |
| Q229 | P16 | *"What were reasons that you wanted to shift to an Agile way of working? "* |
| Q230 | P16 | *"Would you have made that decision if it was not required from upper management? "* |
| Q231 | P16 | *"What are some of the major challenges when implementing an Agile way of working? "* |
| Q232 | P16 | *"Imagine that you get a new developer in the team. What do you want to teach him to allow him to integrate in the team as soon as possible? "* |
| Q233 | P16 | *"Do you think that the Scrum events help in doing that? "* |
| Q234 | P16 | *"When implementing an Agile way of working, are there aspects you focus more on than others because they are more important? "* |
| Q235 | P16 | *"So about the question about being cheaper, did you already came up with something? "* |
| Q236 | P17 | *"What is code quality and how do you make sure that it is applied in the code? "* |
| Q237 | P17 | *"If you would compare those three factors, which do you think is most important? "* |

| | | |
|---|---|---|
| Q238 | P17 | *"What does maintainability mean for you? "* |
| Q239 | P17 | *"Are the other factors that can influence your code quality? "* |
| Q240 | P17 | *"Are there external factors like management that can influence that? "* |
| Q241 | P17 | *"Looking at team dynamics, how does that develop over time in a project? "* |
| Q242 | P17 | *"Are you working co-located? "* |
| Q243 | P17 | *"Do you prefer working in solitude or together? "* |
| Q244 | P17 | *"How long have you been working together? "* |
| Q245 | P17 | *"Why were your attempts in implementing Scrum unsuccessful? "* |
| Q246 | P17 | *"Did you ever feel that you had to do repetitive work for a long amount of time? "* |
| Q247 | P17 | *"Does repetitive work influence your code quality? "* |
| Q248 | P17 | *"Would you like to add anything to any questions I asked? "* |
| Q249 | P18 | *"Can you give a brief summary of Agile and what it means for you? "* |
| Q250 | P18 | *"So Agile is not always the answer. What are disadvantages? "* |
| Q251 | P18 | *"What are advantages Agile offers? "* |
| Q252 | P18 | *"So they get the product faster, is it also cheaper? "* |
| Q253 | P18 | *"How do you create that Agile awareness? "* |
| Q254 | P18 | *"Are there Agile aspects that are more important than others when you are coaching? "* |
| Q255 | P19 | *"What is the Agile way of working and what does it mean for you? "* |
| Q256 | P19 | *"Can you think of disadvantages of the Agile way of working? "* |
| Q257 | P19 | *"Imagine that you overcome all those disadvantages, what are advantages Agile offers? "* |
| Q258 | P19 | *"What causes that efficiency? "* |
| Q259 | P19 | *"What setting do you mean which you cannot improve? "* |
| Q260 | P19 | *"What are the effects deadlines cause in teams? "* |
| Q261 | P19 | *"Does Agile produce pressure for developers? "* |
| Q262 | P19 | *"As a Scrum master, do you focus on anything particular when coaching? "* |
| Q263 | P19 | *"Did you ever experience that a customer had wished the system was documented better afterward? "* |

| | | |
|---|---|---|
| Q264 | P19 | *"How could you have prevented that? "* |
| Q265 | P20 | *"What is your definition of code quality? "* |
| Q266 | P20 | *"Why is it important? "* |
| Q267 | P20 | *"What causes bugs in the system? "* |
| Q268 | P20 | *"How would you solve those problems, or even prevent them? "* |
| Q269 | P20 | *"Does it happen that a tester is not included in the process? "* |
| Q270 | P20 | *"How can you raise awareness that the customer learns the value of Agile? "* |
| Q271 | P20 | *"When working Agile, what is important in the team? "* |
| Q272 | P20 | *"Do you think that a change in processes is inevitable in development teams? "* |
| Q273 | P20 | *"How do the processes change over time? "* |

# Appendix E

# Key Quotations

In this appendix we give some of the most important quotations of participants. Each quotation is given together with the selection of codes we used to tag them and which ID these have in Appendix C. The transcript were originally in Dutch so all quotations have been translated.

Table E.1: Key Quotations of P1

| | | |
|---|---|---|
| 15, 29 | Maintainability, Understanding the Code | *"My definition of code quality is that it is easy to follow by developers. [...] An other aspect is how easy it is to make modifications."* |
| 14 | Keep Learning and Improving Code | *"I always tell myself that I write bad code. Then you remain sharp and then you pick up new stuff."* |
| 13, 16 | Iteration Pressure, Minimise Pressure | *"It depends on the Scrum environment. If it is pushy because work has to be completed, code is written more messy under that time pressure."* |
| 21 | Team Agreements | *"It is important that, together with your team and other teams if necessary, that you work together with them to agree on a code structure."* |
| 19 | Peer Feedback | *"It is useful when someone else looks at your work because he looks at it from a different perspective. Additionally, he has not read what it does yet which makes it easier to spot mistakes."* |
| 31 | Work Floor Discussions | *"Then it is easier to to stand together, grab a whiteboard and sketch it out."* |

Table E.2: Key Quotations of P2

| | | |
|---|---|---|
| 5, 17 | Code Quality Saves Time, Non-functional Awareness | *"I have a strong conviction that good quality is cheaper than bad quality. And sometimes you have to explain why, especially in software development. Our sales department should start with making that clear to the customer."* |
| 18 | Open Communication | *"That requires much more than only diplomacy and sensitivity to deal with it. To recognise it, to acknowledge it and to deal with it, that is difficult for teams."* |
| 17 | Non-Functional Awareness | *"I think that is begins with awareness. So everyone has to be aware that the things you do affect the quality of the final product. [...] So awareness that everyone, including the customer and everyone in the process, understands that quality is important. "* |

Table E.3: Key Quotations of P3

| | | |
|---|---|---|
| 29 | Understanding the Code | *"The code should be easy to read and explore."* |
| 14 | Keep Learning and Improving the Code | *"It works well, but you have to improve it, make it better, smarter, in the next iterations."* |
| 24 | Team Agreements | *"In the beginning you establish some rules, best practices, coding guidelines. And discuss what is useful and what is not."* |
| 19, 31 | Peer Feedback, Work Floor Discussions | *"It is important that everyone is sitting together. Often you have a small question which you cannot post online and wait for an answer."* |
| 16 | Minimise Pressure | *"Deadlines are the nemesis of quality, and personal growth."* |

Table E.4: Key Quotations of P4

| | | |
|---|---|---|
| 15, 29 | Maintainability, Understanding the Code | *"The most important aspects of code quality is that it is easy to read and that it is easy to maintain."* |
| 7, 19 | Continuous Feedback, Peer Feedback | *"I think that a review process is important. That multiple people always review your code."* |

| | | |
|---|---|---|
| 18, 23, 25 | Open Communication, TD Management, Team Openness | *"You should always do technical debt. That is what makes the difference between quality that is good and bad. [...] And keep some sort of Technical Debt backlog. And make sure that in an Agile team that your Product Owner, Scrum Master and the person responsible for the budget agree with it and that you are given time to work on it."* |
| 14 | Keep Learning and Improving the Code | *"You look at code incidents, look what went wrong and try to learn something from it for the future."* |
| 17, 22 | Non-Functional Awareness, Splitting Complex Tasks from Sprint | *"A disadvantage of Agile can be when you have short iterations, and you have not thought about the total product like the architecture, you will get increments with a scope on which you keep building. Eventually you will get a crooked building."* |

Table E.5: Key Quotations of P5

| | | |
|---|---|---|
| 9 | Feeling Comfortable in the Team | *"It is about giving everyone a safe feeling that it is good to get feedback and to do something with it."* |
| 16, 18 | Minimise Pressure, Open Communication | *"I have seen some projects where the pressure became too high and you get the feeling that you cannot say everything anymore. Then it becomes more of a blaming game than trying to find solutions together. And that consumes time without improving anything."* |
| 24 | Team Openness | *"If someone gets the feeling that something is impossible to do he should be able to indicate it and discuss it."* |

Table E.6: Key Quotations of P6

| | | |
|---|---|---|
| 20, 29 | Programmer Efficiency, Understanding the Code | *"I think that code quality is making sure that programmers can continue rapidly on the existing code. Readability is an important aspect."* |
| 7, 12, 19 | Continuous Feedback, Involving All Parties in Development, Peer Feedback | *"Direct Communication between the developers and the end users is beneficial because you forget less and reduce miscommunications."* |
| 17 | Non-Functional Awareness | *"If you include end users in the development, regardless of the actual influence they have, they will accept the choices more easily."* |

| | | |
|---|---|---|
| 25 | Team Empowerment | *"We also had a PO who took the right decisions which allowed us to create a product that was enthusiastically received."* |
| 18 | Open Communication | *"Good communication is important to prevent bugs."* |

Table E.7: Key Quotations of P7

| | | |
|---|---|---|
| 5, 16 | Code Quality Saves Time, Minimise Pressure | *"Often projects are built around deadlines. Code that is written is not thoroughly tested. What I have seen is that you have to do that in the development phase because it is more expensive to modify it in later phase."* |
| 29 | Understanding the Code | *"Code documentation is necessary. And also references. If the software does not allow that, write down the requirement that is linked to the code."* |
| 26 | Team Openness | *"And you have to step up to say that you do not know something."* |

Table E.8: Key Quotations of P8

| | | |
|---|---|---|
| 7 | Continuous Feedback | *"And that the user feedback is immediately taken into the next iterations."* |
| 12 | Involving All Parties in Development | *"Because during the demos of the Scrum process the customer is actively involved. As a result the customer sees you more often where he can shift more of the business to you."* |
| 1, 17 | Agile Awareness, Non-Functional Awareness | *"So it is important that you do some expectation management with you customer. Similar to a house, a lot has to be done that you do not see but it will make sure it will still stand in a couple of years."* |
| 13 | Iteration Pressure | *"And what you see is that developers continuously estimate a little too much, promise a little too much. Which makes them stressed. "* |
| 18 | Open Communication | *"I prefer being able to explain to the customer, we did not expect this, we have to delay it."* |
| 2 | Balance Between Focus and Communication | *"Agile might make the process a little slower caused by all the communication. Every day I lose my focus because I have to tell what I am going to do. Then I need a long time before I reach the same level of focus again."* |

Table E.9: Key Quotations of P9

| | | |
|---|---|---|
| 11 | Flexibility | *"We do not really do Scrum. We do the daily standups and stuff, but at small projects you lose too much time if you spend 1 or 2 days per sprint."* |
| 15 | Maintainability | *"Thus maintainability is a major aspect, maybe even the most important one."* |
| 14 | Keep Learning and Improving the Code | *"At the moment that you write code but do not maintain it, and something has to changed you have a serious problem."* |
| 13, 23 | Iteration Pressure, TD Management | *'At the end of a sprint you suddenly get a new ticket and then you have to do a quick and dirty fix to implement it. Then, the dirty fix is forgotten."* |
| 11, 28 | Flexibility, Team Structure | *"It is a structure that you have to keep without looking at what is working and not in the team. In that sense, Agile might break more than it solves."* |
| 22 | Splitting Complex Tasks from Sprint | *"Scrum does not account for requirement analysis, business analysis, architecture and stuff like that. [...] If you only take Scrum as software development it might work."* |

Table E.10: Key Quotations of P10

| | | |
|---|---|---|
| 7, 12 | Continuous Feedback, Involving All Parties in Development | *"For me Agile is keeping a close connection with the business and end users. [...] Regularly checking, every day, where you can, adjust solutions based on where the project is moving."* |
| 5, 17 | Code Quality Saved Time, Non-Functional Awareness | *"It is extremely focused on functionality. If the system performs badly, is not secure or not maintainable, we let it pass because in the end we have delivered the required functionality which makes the Product Owner satisfied. The rest they only see when the system breaks down, and then it is too late."* |
| 11, 24 | Flexibility, Team Structure | *"A think that a framework definitely adds value, but the art is to use it at the proper moments. "* |
| 1 | Agile Awareness | *"I mostly try to raise awareness and train and coach people in that."* |

Table E.11: Key Quotations of P11

| | | |
|---|---|---|
| 29 | Understanding the Code | *"First of all documentation. That is extremely important. There are plenty application that are built from a model in your head. But that you understand it does not mean I understand it."* |
| 1 | Agile Awareness | *"Explain the value of Scrum. That it is important for us and that we achieve better results using it."* |
| 22 | Splitting Complex Tasks from Sprint | *"What I have learnt is that you should map your requirements way in advance and that you must map what is in scope and what is out of scope."* |

**P12**

Due to an audio-technical issue we were unable to record this interview. We used our notes to write memos.

Table E.12: Key Quotations of P13

| | | |
|---|---|---|
| 28 | Team Structure | *"Because you impose some discipline at given times, you impose retros, reflections. A fixed planning imposes predictability in the system, which is nice for a new developer without much experience, or experienced developers who are new in the team."* |
| 11 | Flexibility | *"We do not apply Scrum 100%. We more or less use the stuff we can use, and do not use things that limit us. "* |
| 7, 28 | Commit to be Responsible, Team structure | *"You have responsibility during your review. In that way you give developers more responsibility, possibilities to grow and more structure to work in. As a result, you can deliver software with some expectations and release schedules."* |
| 18, 25 | Open Communication, Team Openness | *"He will most likely get a reprimand. That is a direct result from the fact that the Scrum Master did not keep a close eye on the team to spot something went wrong in the team."* |
| 1, 8 | Agile Awareness, Culture | *"It is one of the things that is always difficult. To have non-technical people take control over technical things. Unfortunately, sales and product managers often promise things that cannot be done in the budget or time. So there should always be technical person in those processes."* |
| 13, 26 | Iteration Pressure, Team Openness | *"And then there comes a time that you are swamped by tasks with the note that you only have to do it. Then you should step up and say that you cannot do this."* |

Table E.13: Key Quotations of P14

| | | |
|---|---|---|
| 11 | Flexibility | *"It can go wrong with Scrum if you do not use it properly or take all the rules too seriously."* |
| 26 | Team Openness | *"And that you have discussions in the team that everyone knows what the team is doing which makes it easier to distribute tasks."* |
| 1 | Agile Awareness | *"And I think that people people should not follow the Scrum guide completely but use what best suits the business."* |
| 24 | Team Agreements | *"I think that if you properly implement Scrum it has a positive effect because if you maintain the definition of done then everything will have a high quality."* |

Table E.14: Key Quotations of P15

| | | |
|---|---|---|
| 27 | Team Predictability | *"It increases the efficiency because you have the tasks, the short meetings, and you know exactly what you must do so you can prepare for them."* |
| 11, 21 | Flexibility, Self-Organising Teams | *"If the team has some knowledge of Scrum then it works. It depends on the team. If something does not work than you must change that to make it work."* |
| 1 | Agile Awareness | *"It is not the responsibility of the team to remove that impediment. But if the team is not aware of that it can go wrong. I think that is a disadvantage. You must give courses to the team to use it properly."* |
| 2 | Balance Between Focus and Communication | *"I have had a meeting that took way to long. That should not happen in Scrum. You should time your meetings and keep them as short as possible and make sure people know what they should know before joining the meeting."* |
| 11 | Flexibility | *"I think that it is useful to adapt a little to the needs. I think that Scrum agrees with that, that you can be adaptive."* |

Table E.15: Key Quotations of P16

| | | |
|---|---|---|
| 12 | Involving All Parties in Development | *"What Agile should add for me is that it is more about the interaction"* |
| 7 | Continuous Feedback | *"I now have experience with the intermediate demos of small pieces. And I think that it is very valuable. That the customer orders something and that we are able to build a first draft and show it as working software."* |
| 11 | Flexibility | *"At which point we do not try to cling to all the do's and don'ts of Scrum Agile. So we try to give it our own twist"* |
| 8 | Culture | *"The Agile Scrum way of working also deals with culture. That is an important aspect and not everyone does this naturally. What you often see is, especially old school developers, is that they want a piece of paper with a functional design with what you want. And they will build it for you. [...] As far as I'm concerned, Agile means working together. Preferably co located."* |
| 6, 21 | Commit to be Responsible, Self-Organising Teams | *"Instead, we are responsible as a team. And we will do this as a team. So if the tester is sick and there is some testing that has to be done, we will do that for the tester. So we do not wait until the project manager gives us a new tester. Team responsibility, that really has to do with culture of the people, but also with the environment."* |
| 1 | Agile Awareness | *"Maybe this belongs to culture, but a little acceptance of the role of the Product Owner, who is the boss of the Agile team. It has to be accepted and supported by higher management. [...] By giving a mandate, especially to the Product Owner. And get the right people in your teams."* |
| 26 | Team Openness | *"You must have the guts to say what goes wrong in a team. And that can be about people. But you cannot let that fester in a team."* |
| 9 | Feeling Comfortable in the Team | *"Do an intake. The team has to work together with him, so let the team decide if it is a match."* |

Table E.16: Key Quotations of P17

| | | |
|---|---|---|
| 15, 29 | Maintainability, Understanding the Code | *"My opinion about quality is that it is clearly structured and easy to maintain."* |
| 26 | Team Openness | *"And we have one person that always wants to please the users. Then I am the person that slows us down or else we have to work overtime."* |
| 14 | Keep Learning and Improving the Code | *"We have someone in our team who does not write maintainable code. And we know that. So when he builds something we make it better and more maintainable."* |

Table E.17: Key Quotations of P18

| | | |
|---|---|---|
| 1, 8 | Agile Awareness, Culture | *"If a business does not have a culture to delegate for example, so not able to give some mandate to the product owner. Then part of your Agile processes will not work because the Product Owner cannot make his own decisions."* |
| 11 | Flexibility | *"So for each project the client implements its own Agile processes."* |
| 12 | Involving all Parties in Development | *"It provides a way where you achieve a partnership with the client. This way you can collaborate very efficiently. Consequently, the we-they culture that you often see in fixed price projects and specification contracts, diminishes."* |
| 7 | Continuous Feedback | *"I think, because you constantly validate if you are heading in the right direction, the client gives his insights on the product much easier. This allows you to make modifications way faster which you would have otherwise encountered at the final delivery of the product."* |
| 26 | Team Openness | *"It is better to have a Product Owner who knows what he does. And if that is not the case, then you must help him in that and start operating as a team. That is a project."* |
| 1, 21, 28 | Agile Awareness, Self-Organising Teams, Team Structure | *"I would start with the leadership. So the management that stands above it. Because they have to give trust to those self-organising teams, hence the culture of Agile and openness and constantly wanting to learn."* |

| | | |
|---|---|---|
| 16 | Minimise Pressure | *"The responsibility of tasks that lie outside the team should not be dropped in the team for them to solve."* |
| 8, 9 | Culture, Feeling Comfortable in the Team | *"I think culture, way of working cooperatively, in the sense of how do we work together? How do we do it together? It is extremely important to give a team a goal. And not a goal like build a project but to give a common goal like at the end of the project this is what we want to achieve, together."* |
| 1 | Why Agile > Doing Agile | *"At the moment an organisation wants to do all their software development in an Agile way, do not implement it organisation-wide, but start slowly. A customer has to grow in the Agile process and do not expect from the customer that he does it perfectly from the first day. I think that complying with the Agile values starts at the customer. Few customers are transparent and open in trust. So you have to guide them in that and the best way to do that is to follow them yourself."* |

Table E.18: Key Quotations of P19

| | | |
|---|---|---|
| 21 | Self-Organising Teams | *"For me, Agile means listening to your developers because they possess all the knowledge. In the end, they build what the customer wants. And the adaptive policy, which is constantly adapting, and focus on that, that is Agile."* |
| 30 | Why Agile > Doing Agile | *"And those are all potential disadvantages caused by a bad execution of Agile."* |
| 9 | Involving All Parties in Development | *"The advantage is that the client gets what he actually wants. You get less problems between developers and the Product Owner. And that is basically caused by the intensive communication. It happens almost automatically."* |
| 20 | Programmer Efficiency | *"And then you wait if it is better and if it is not you change again. That causes that your efficiency increases. If you work with the right team. And in a different setup there is no incentive to improve or to be more efficient, you simply miss it."* |

| | | |
|---|---|---|
| 6, 16 | Commit to be Responsible, Minimise Pressure | *"The harder you push developers to make a deadline, they will reach it. However, you must not ask with what kind of code, or with a lack of tests. Things are pushed to reach the deadlines and you see that a lot. And Agile removes it by basically allowing to start over each sprint. With Agile you commit as a team to those 2 weeks and I think that is powerful. That deadlines are not imposed, but that you accept it, or create it."* |
| 4, 8, 13 | Challenge Yourself, Culture, Iteration Pressure | *"Perhaps inside the sprint, if you committed as a team to reach the sprint goal. Then you have the pride as a team to deliver that. While the Agile culture states that when you face problems you must discuss them with your Product Owner and Scrum Master, but that is a risk. You have to keep challenging the sprint deadline."* |
| 17 | Non-Functional Awareness | *"I think customers or Product Owners often underestimate that. As a user I want to have this and this because I work more easily then. They think that it is clear enough and we have to understand it."* |
| 19 | Peer Feedback | *"Placing the responsibility with the team. And stimulate the team to give feedback."* |
| 1, 17 | Agile Awareness, Non-Functional Awareness | *"By thinking earlier how you add that part to your definition of done. You have the definition of ready which you should have with your requirements, and especially the definition of done. But I think, the expectation management, how much time it takes to have your system properly documented, that is an aspect for customers. Although is is an investment for the future. When you work on a piece of functionality everyone knows that you are building. however, this knowledge decreases sprint after sprint. "* |

Table E.19: Key Quotations of P20

| | | |
|---|---|---|
| 7, 19 | Continuous Feedback, Peer Feedback | *"If bugs emerge it costs more time to fix them then, for example, when you developed it in an Agile way where you get feedback from the tester if something is working or not during every sprint feedback."* |
| 1 | Agile Awareness | *"So if you can translate the advantages of Agile development to fewer costs, lower costs in software development, or lower costs because they have to pay less compensation if something goes wrong. [...] If you want to convince an organisation to go Agile I would make a cost calculation or you must run a trial project where you show what you can deliver in a short amount of time."* |
| 18, 19, 31 | Open Communication, Peer Feedback, Work Floor Discussions | *"I think that direct communication is very important. That you sit together, interact, and that you can have discussions with each other."* |

# Appendix F

# Questionnaire

# Code Quality in Agile Software Development

This questionnaire contains 29 closed and 7 open questions questions, divided in 6 sections. It will take approximately 15 minutes to complete.

Thank you for your interest in our research on Code Quality in Agile Software Development. In this questionnaire we are interested in the influence of Agile on your ability to write high-quality code in your daily routine. You will be given statements for which you can indicate how strongly you disagree or agree. If you have a neutral opinion, please select the center option (value 3).

We will ask questions about your development team. This includes the Product Owner and Scrum Master if you use Scrum.

We are interested in the opinions of developers or people who used to develop.

First, we will need some information to get started.

*Required

1. **1. Which country do you live in?** *
   *Mark only one oval.*
   - ◯ Select Country ...

2. **2. Please specify your gender**
   *Mark only one oval.*
   - ◯ Male
   - ◯ Female
   - ◯ Other: _____

3. **3. What describes your role best?** *
   *Mark only one oval.*
   - ◯ Full-time developer
   - ◯ Part-time developer
   - ◯ Student
   - ◯ Used to be a developer
   - ◯ Other: _____

4. **4. Please select which Agile implementations you have worked with**
   *Tick all that apply.*

   - [ ] Crystal
   - [ ] Dynamic Systems Development Method (DSDM)
   - [ ] Extreme Programming (XP)
   - [ ] Feature-Driven Development (FDD)
   - [ ] Kanban
   - [ ] Lean
   - [ ] Scrum
   - [ ] Other: _____

5. **5. Please select how many years you have worked with Agile** *
   *Mark only one oval.*

   - ◯ No experience
   - ◯ < 1 year
   - ◯ 1-2 years
   - ◯ 3-5 years
   - ◯ 6-10 years
   - ◯ > 10 years

## Development Pressure

The following statements are related to pressure in the development environment.

Please indicate how strongly you (dis)agree with the following statements.

6. **6. I write better code when I am not facing too much pressure** *
   *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

7. **7. I feel Agile software development allows the development team to operate more independently** *
   *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

8. **8. Being part of an independent development team reduces unnecessary pressure** *
   *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

9. **9. I feel Agile software development allows me to work in a sustainable pace** *
   *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

10. **10. Agile software development improves the communication of our development team** *
    *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

11. **11. I feel being open and honest in the development team can improve / improves our problem solving process** *
    *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

12. **12. For one or more of the above statements, please specify the question number and why you feel that way.** *

## Awareness

The following statements are about awareness of the development process e.g. software methodology, expectation management, code quality, maintainability, documentation, security etc.

Please indicate how strongly you (dis)agree with the following statements.

13. **13. I think that the members of my development team should be enabled to fulfill their responsibilities** *
    *Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

14. **14. I feel people outside our development team have influence over important decisions in the project** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

15. **15. I feel (technical) decisions made by our team are respected and accepted by people outside of the development team** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

16. **16. Agile software development increases the transparency of our development (team)** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

17. **17. Agile software development increases the efficiency of our development team** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

18. **18. For one of the above statements, please specify the question number and why you feel that way.** *

## Team Culture

In the following section we query statements about how team members think and interact with each other.

Please indicate how strongly you (dis)agree with the following statements.

19. **19. People of our development team help each other when they get stuck** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

20. **20. My development team gives feedback on all aspects of each others work** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

21. **21. When software development problems are identified, my development team improves the processes if needed** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

22. **22. I feel everyone is included in the decision-making processes of my development team** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

23. **23. Feedback from the development team improves my development skills and the quality of my code** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

24. **24. I think the development team should have a common goal** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

25. **25. For one of the above statements, please specify the question number and why you feel that way.** *

## Team Processes

The following statements are related to the processes your team follows and the agreements they made.

Please indicate how strongly you (dis)agree with the following statements.

26. **26. I think agile software development makes work more organised** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

27. **27. I feel creating clear (flexible) agreements help in following the processes of the team structure**
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

28. **28. I think culture plays an important role in establishing the processes** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

29. **29. I feel the team keeps development processes that work well** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

30. **30. Agile software development increases my development speed** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

31. **31. I think Agile software development makes my work more fun** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

32. **32. I prefer an Agile approach over the Waterfall model** *
*Mark only one oval.*

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ◯ | ◯ | ◯ | ◯ | ◯ | Strongly Agree |

33. **33. For one of the above statements, please specify the question number and why you feel that way.** *

## Open Questions (optional)

The following questions are optional. This is the last section. The form will be submitted after this page.

34. **34. What do you consider as the main benefit(s) of the agile implementation in your organisation?**

35. **35. What do you consider as the main challenge(s) of the agile implementation in your organisation?**

36. **36. If you could change one thing, what would you change about the agile process you use(d)?**

# Appendix G

# Agile Software Development Best Practices Poster

# Improving Code Quality in Agile Software Development

Insights into how you can improve your team's code quality metrics through enhancing your Agile processes

## Version Control Systems

A Version Control System gives another means of documentation. Design choices can more easily be tracked. It also provides a way to work faster together through branching, for example.

## Refactoring

If the code is not as good as it should be, take time to refactor some parts of the code that is frequently worked on. Not only will this save time in the future because developers can add features more easily, it also stimulates developers to write high-quality code in these parts.

## Definition of Done

Define rules when code is considered to be Done and ready to be merged to production. Include quality rules in the Definition of Done.

## Team Agreements

You can make agreements with coding standards or practices your team will use to enhance code quality. Implement measures to check that code satisfies the agreements.

## Eliminate Technical Debt

All Technical Debt that is created should be accounted for. For example, if you have to take a shortcut to save some time, add the task that has to be done to properly implement that shortcut as Technical Debt and estimate how long it will take, instead of adding a simple TODO in comments. Take measures to be able to work on Technical Debt.


## Involve All Parties in Development

Parties that have an interest in the product should be involved in the development. Developers are not the end-users of the system and might not have all the information to perfectly make the product. Allowing parties involved to provide feedback gives developers a better picture of what has to be done, has gone wrong or could be done. It enables discussions between technical and non-technical staff which makes it easier to explain technical decisions.

## Direct Communication

Communication between team members is direct. Most communication should go via a face-to-face discussion or Skype/phone-calls. Communication where you have to wait for an answer should be minimal. This decreases the time developers are stuck and reduces misinterpretations.

## Feedback

Feedback can be between developers or on an individual level e.g. analysis tools. Enable developers to give feedback on each other's way of working and written code. Introduce tools that help developers in following standards or to find possible flaws in the code.

## Continuous Integration

Introduce Continuous Integration in your development pipeline to automatically run tests, check on your coding guidelines and practices and to verify if your code builds.

**DISCLAIMER**

**You should decide what and how to change because we do not know where you are or where you are going. We do not know what you should do because we do not know what happens next. You should find out where you are, what you need and how you should adjust. This poster should only be used to gain insights into how you can change.**