Markus Kammerhofer, BSc.

Enabling Project Success in a Very Small Web Company

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl-Ing. Dr.techn. Franz Wotawa

Co-Supervisor

Dipl-Ing. Dr.techn. Bernhard Peischl

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Graz, May 2019

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date	Signature

Abstract

Very small software companies have a significant economic impact. As these companies are most often cash flow driven, project success is vital for them. In the context of small, medium and large enterprises, software process improvement is already proved to enable project success. Unfortunately, very small companies differ even from small companies decisively. Therefore the common belief of very small companies that software process improvement initiatives cause an unaffordable overhead is yet not empirically negated. This work aims to prove the opposite by conducting an empirical action study to a very small web company in Austria. Over 14 months two iterations of the Quality Improvement Plan — an iterative light-weight software process improvement framework — were executed. In each iteration, goals were defined and quantitatively evaluated by using a customized quality model, which was iteratively developed by using the Goal Question Metric approach. After analyzing a project, appropriate corrective actions were chosen and applied to the subsequent project. Ultimately the impact of these actions was evaluated quantitatively by the quality model. With this methodology, systematic requirements engineering and modern code reviews were integrated into the development process in the first iteration, and continuous delivery was integrated in the second iteration. The results show that software process improvement initiatives do not need unaffordable overhead. By conducting the light-weight initiative along with the three corrective actions, the projects stuck to budget and time constraints more accurately. In addition to that, the number of failures found by customers, and the lead and cycle times decreased significantly. Furthermore, a vast improvement of the quality of the code bases could be achieved.

Abstract

Kleinstunternehmen haben einen starken Einfluss auf die Wirtschaft. Nachdem diese Unternehmen zumeist stark von ihrem Cashflow abhängig sind, ist der Erfolg von Projekten überlebenswichtig. Für kleine, mittlere und große Unternehmen wurde bereits gezeigt, dass Prozessverbesserungsinitiativen einen positiven Einfluss auf den Projekterfolg haben. Kleinstunternehmen unterscheiden sich jedoch sogar von Kleinunternehmen sehr stark. Die Befürchtung, dass diese Initiativen für Kleinstunternehmen einen unleistbaren Mehraufwand bedeuten, konnte bisher noch nicht ausgeräumt werden. Das Ziel dieser Arbeit ist es mittels einer empirischen Studie in einem österreichischen Kleinstunternehmen aus dem Webentwicklungsbereich, das Gegenteil zu beweisen. Über einen Zeitraum von 14 Monaten wurden zwei Iterationen des Quality Improvement Plan's — ein iteratives und schlankes Prozessverbesserungsframework — durchgeführt und aktiv begleitet. In jeder Iteration wurden Ziele definiert, deren Erreichen dann auch quantitativ mittels eines Qualitätsmodells überprüft wurden. Dieses Qualitätsmodell wurde iterativ mit dem Goal-Question-Metric-Ansatz erstellt. Nach der Analyse jedes Projekts wurde der Prozess angepasst. Letztendlich wurden die Wirksamkeit der Änderungen und das Erreichen der Ziele der darauffolgenden Projekte mit dem Qualitätsmodell festgestellt. Auf diese Weise wurden eine systematische Anforderungsentwicklung, moderne Code Reviews und Continuous Delivery Schritt für Schritt in den Entwicklungsprozess integriert. Die Ergebnisse zeigen, dass Verbesserungsinitiativen selbst für Kleinstunternehmen keinen unleistbaren Mehraufwand bedeuten. Die Mehrkosten wurden durch die folglich finanziell erfolgreicheren Projekte abgedeckt. Neben der besseren Einhaltung von Zeit- und Budgetzielen wurden auch wesentlich weniger Fehler von Kunden gefunden. Darüber hinaus wurden die Durchlaufzeiten und die Codequalität stark verbessert.

Contents

1 Introduction **1**

2 Description of Problem and Solution Approach **5**
2.1 Description of Investigated Company 5
2.1.1 Manpower Analysis 5
2.1.2 Analysis of Process and Structure 8
2.1.3 Drawbacks 12
2.2 Elaboration of Approach to enable Project Success 14
2.2.1 Project, Project Management and Project Success 14
2.2.2 Enabling Project Success 17
2.2.3 Software Process Improvement 18
2.3 Approach 20
2.3.1 SPI Frameworks and Properties 21
2.3.2 Situation in Very Small Enterprises 31
2.3.3 Selection 33

3 Implementation **37**
3.1 Iteration 1 37
3.1.1 Description of Project A 38
3.1.2 Observations of Project A 38
3.1.3 Define Goals 41
3.1.4 Initialize GQM-Model 43
3.1.5 Quantify Observations of Project A 52
3.1.6 Description of Project B 58
3.1.7 Corrective Actions 58
3.1.8 Analyze Impact on Project B 64
3.2 Iteration 2 71
3.2.1 Observations of Project B 72
3.2.2 Define Goals 74

3.2.3 Extend GQM-Model 74
3.2.4 Quantify Observations of Project B 79
3.2.5 Description of Project C 85
3.2.6 Corrective Actions 86
3.2.7 Analyze Impact on Project C 91

4 Related Work **105**
4.1 Classification Framework 105
4.2 Inclusions and Exclusions 107
4.3 Publications 108

5 Conclusions **115**
5.1 Recap 115
5.2 Discussion of SPI Approach 118
5.3 Discussion of Applied Practices and Threats to Validity 120

Bibliography **123**

List of Figures

2.1 Number of employees 6
2.2 Average hours worked 7
2.3 Total hours worked 8
2.4 Team structure in the beginning. Different sizes of software engineers related to different amount of working-hours per week. . . . 9
2.5 Current team structure. Different sizes of software engineers related to different amount of working-hours per week. 11
2.6 The Iron Triangle [4] 15
2.7 Categorization of CSFs [20] 19
2.8 Improvement goals [77] 20
2.9 Types of progression 23
2.10 The IDEAL model [56] 24
2.11 CMMI v1.3 [32] 28
2.12 Relationship of ISO 15503, ISO 12207 and ISO 9001 [32] 30
2.13 GQM Approach [82] 32
2.14 Quality Improvement Plan [82] 34

3.1 Dependencies of observations 42
3.2 Project A: Effort distribution 54
3.3 Project A: Distribution of lines of code on files (7185 total lines of code distributed on 43 files) 55
3.4 Project A: Distribution of cyclomatic complexity on files (Total complexity of 1690 distributed on 43 files) 55
3.5 Project risk framework [45] 60
3.6 Elaborated process 63
3.7 Project duration 65
3.8 Project effort 65
3.9 Project B: Additional effort and effort distribution 65
3.10 Project A: Effort distribution 66

3.11 Project B: Effort distribution 66
3.12 Tree map of lines of code of projects A and B 70
3.13 Tree map of cyclomatic complexity of projects A and B 71
3.14 Lead and cycle times of project B 83
3.15 Project A: Effort distribution until due day 85
3.16 Project B: Effort distribution until due day 85
3.17 Process adjustments for iteration 2 (Changes presented in black) . 90
3.18 Tree map of lines of code of projects A, B and C 94
3.19 Tree map of cyclomatic complexity of projects A, B, and C 95
3.20 Lead and cycle times of projects B and C 99

4.1 Classification framework 106

List of Tables

2.1 Square route for understanding success criteria [4] 17

3.1 Observations of project A 41
3.2 G1 prevent scope creep: Measurement results project A 53
3.3 G2 deliver maintainable code: Measurement results project A . . . 57
3.4 G1 prevent scope creep: Measurement results project A vs. project B 67
3.5 G2 deliver maintainable code: Measurement results project A vs. project B 69
3.6 Observations of project B 74
3.7 G3 achieve a schedule estimation accuracy of 1: Results of projects A and B 80
3.8 G1 prevent scope creep: Measurement results of projects A, B and C 93
3.9 G2 deliver maintainable code: Measurement results of projects A, B and C 97
3.10 G3 achieve a schedule estimation accuracy of 1: Results of projects A, B and C 100

1 Introduction

This thesis deals with enabling project success in very small enterprises by conducting a light-weight software process improvement initiative to a very small web development company in Austria. In the process, a quality model is generated iteratively, and three common software development practices are established one by another. Finally, most importantly the success of the established practices and the overall software process improvement initiative is quantitatively determined.

The software industry is still one of the fastest growing worldwide. It contributes 2.9 million jobs in the United States and 3.6 million in the European Union in 2016. These figures grew by 14.6 respectively 16.5 percent since 2014 [2][1]. In Austria 5,089 software development companies employ 27,320 including 5,030 self-employed people in 2016. Ninety percent of these companies are very small enterprises with less than ten employees. Another eight and two-tenths percent are small-sized enterprises with 10 to 49 employees. The remainder is spread among medium and large-sized enterprises [5]. These figures are similar to many other countries like the United States, Brazil, Canada, China, India, Finland, Ireland, and Hungary [41]. As one will recognize, these very small companies have a big impact on the economy. In contrast to their large counterparts, small and medium-sized enterprises are most often cash flow driven. Project success is therefore critical to stay competitive and ultimately, to survive [49].

Project success is defined by the iron triangle regarding adherence to budget and time constraints, and delivering an appropriate quality. The quality of a created product has to be verified and validated. It has to be ensured that the delivered product sticks to a specification, the so-called verification, and it has to be validated that the specification and therefore the product serves the intended purpose. Software development companies operating in the service sector have to focus more on verification because the customer often determines the requirements. Therefore the customer carries the ultimate responsibility, and the development company can

only serve as an advisor. In addition to these fundamental success factors, organizational and stakeholder benefits can be considered [4]. Whereas time, cost and quality can be determined at the day of acceptance, organizational and stakeholder benefits can only be evaluated in the post-acceptance phase. Project success can be achieved in many ways, e.g., by improving a team's capability or by improving the processes. A business process consists of defined and structured interrelated tasks that are performed in a specific order to produce a service or a product for customers. These processes occur on all levels of an organization [72]. Most of the literature especially regarding software process improvement was related to large companies. At the beginning of the new millennium research recognized the need for investigation in that direction. This was followed by a high interest in that topic for many years then. However, also these publications were mainly related to small and medium-sized enterprises. This is a problem as Christian Hofer already found that very small companies with less than 10 employees have significant differences to companies with 10 to 50 employees [36]. Also, the agile movement is not fully applicable to very small companies, even if the combination of agility and software process improvement gained interest since 2008 [47]. On the one hand, it is not fully applicable because most of the contracts of very small companies include a fixed price. On the other hand, the average team size is between one and two and a half members [76]. Agile methodologies need larger teams, e.g., Scrum recommends teams consisting of five to six developers. As one can see, research regarding project success in very small companies is most often omitted.

Von Wangenheim and Richardson state that people often believe that good practices and solutions are expensive, time-consuming, and targeted toward large organizations, and therefore difficult to apply in small companies [41]. However, research already showed that software process improvement in small and medium-sized companies does not introduce unacceptable overhead [91][48][59]. However, there is still too little evidence, that it is also profitable for very small companies. Therefore the goal of this work is to provide evidence by conducting an action study in a very small company. To accomplish this goal, it was only concentrated on the most promising process areas during this research. It was also important to evaluate the success of such process improvements. This was achieved by creating a customized quality model using the Goal Question Metric approach iteratively. Therefore a standardized assessment was not used to avoid unnecessary overhead. It is expected, that focusing on the process areas, which cause the most severe problems will improve the project success significantly, according to the Pareto

principle. Also due to the fact, that project success is quantitatively determined for the first time, a more conscious and better decision making is expected. The whole company should, therefore, be better steerable than before.

In chapter 2 the investigated company is described in detail. In addition to that, different ways to enable project success are presented, and one concrete approach is chosen to be used in this context. In chapter 3 the implementation of the iterative software process improvement initiative and its results are described. Two concrete iterations over 14 months including three projects were therefore accompanied. The whole process of observing the most important areas to improve, up to the selection of the corrective actions, and the analysis of their impact, is described comprehensively. In chapter 4 related work is presented. Finally, this thesis is concluded and discussed in chapter 5.

2 Description of Problem and Solution Approach

This chapter will explain the methodology and its derivation in detail. At first, a description of the investigated company is provided. Subsequently, software process improvement as the topic of interest is chosen. The choice was made by having a look on project success and the project success factors. At last a concrete approach is presented.

2.1 Description of Investigated Company

This section provides a comprehensive description of the current status of the company and how it evolved. For analyzing the company's organizational structure, an analysis of the workforce over time is provided. This analysis shows how fast and in which way the company grew. After that, the evolution of the company's organizational structure over time is reproduced. This creates a link to workforce analysis. In the end, the drawbacks of the current situation are carved out.

2.1.1 Manpower Analysis

Three different metrics were chosen to describe the growth and the evolution of the company,

- Total hours worked per month
- Average hours worked per month
- Number of employees per month

The total hours worked per month show the growth of the company. Whereas the average hours worked and the number of employees per month, give more insights about the structural change over time.

Number of employees

At the beginning of the company there was only one individual developer, who founded the company. When the workload got too big, the founder was looking for support by an employee. This procedure was repeated many times. As the effort for managing more employees is more likely to be exponential than linear, recruiting more people over and over again, was not the answer in the long run. Therefore the curve of the number of employees by time flattened out as figure 2.1 illustrates. A process started where the average time worked by the employees increased.

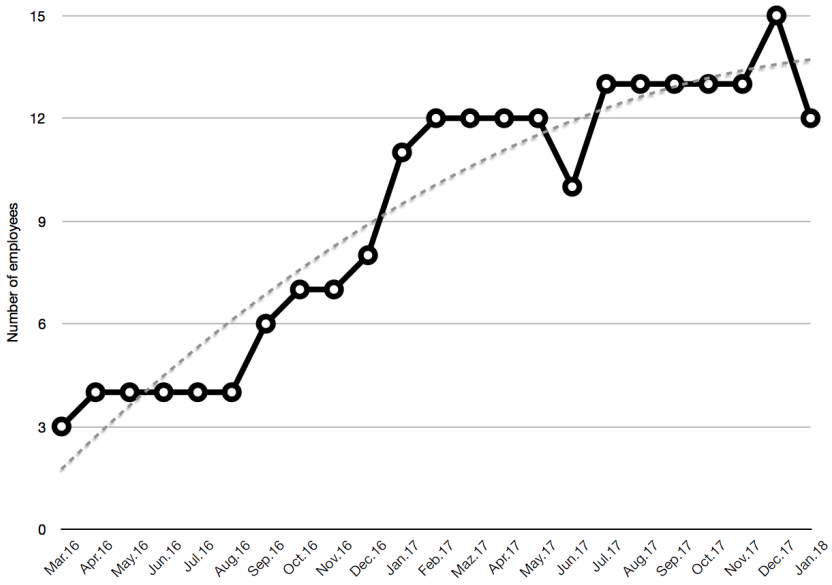

Figure 2.1: Number of employees

Average hours worked

The employed software engineers consisted entirely of part-time students. The reason for that is the founders need of low salary, flexibility and therefore part-time employment because he was not able to guarantee a full-time job. This scheme perfectly fits the needs of students. As already shown, employing more and more students to overcome the workload stopped scaling at one point in time. Increasing the number of employees was not paying off anymore, because the effort of recruiting and integrating new employees was high. Caused by the bigger size of the company and a stable order situation, some full-time developers were acquired, and already employed students increased their amount of work per week continually. Therefore the average hours worked per week increased linearly over time as seen in figure 2.2. The variations of those numbers are caused by the varying effort for the students at the university. The impact of this measurement and the number of employees on the total performed hours within the company is shown and discussed in the following section.

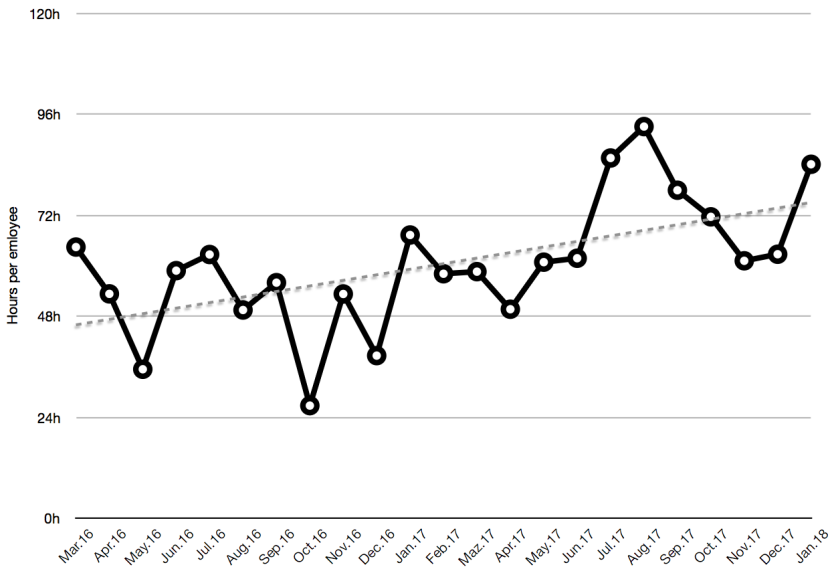

Figure 2.2: Average hours worked

Total hours worked

The consequence of more people working more hours on average led to an exponential growth of the company as shown in figure 2.3. In section 2.1.2 the needed and applied adjustments in the company's organizational structure can be found. Of course this exponential growth caused some kind of chaos, further described in the section drawbacks. The aim of this thesis is to respond to these negative side effects, which are common in small enterprises [41][68].

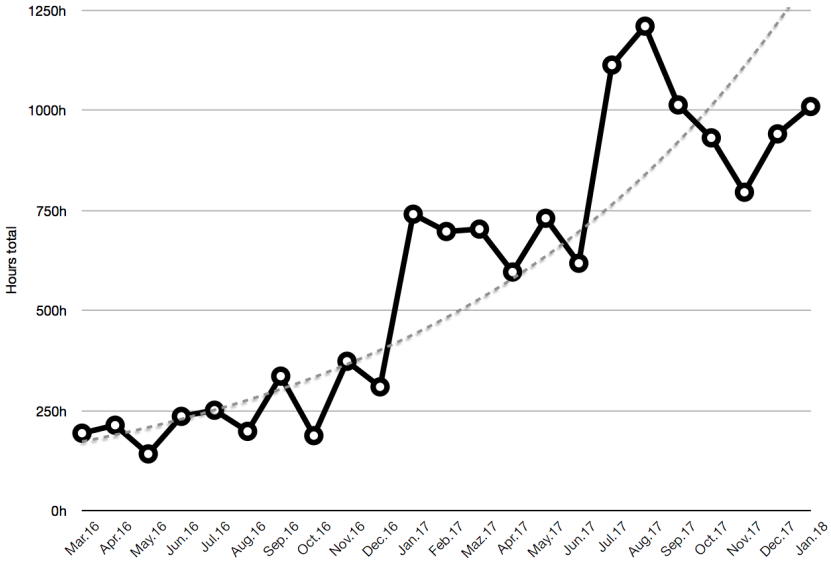

Figure 2.3: Total hours worked

2.1.2 Analysis of Process and Structure

As described in section 2.1.1, the response to the increasing amount of workload was employing new software engineers at first and then increasing their average amount of working hours per week. This growth caused the necessity of adjustments in the organizational structure of the company. The initial approach after acquiring some new employees was a star structure as shown in figure 2.4.

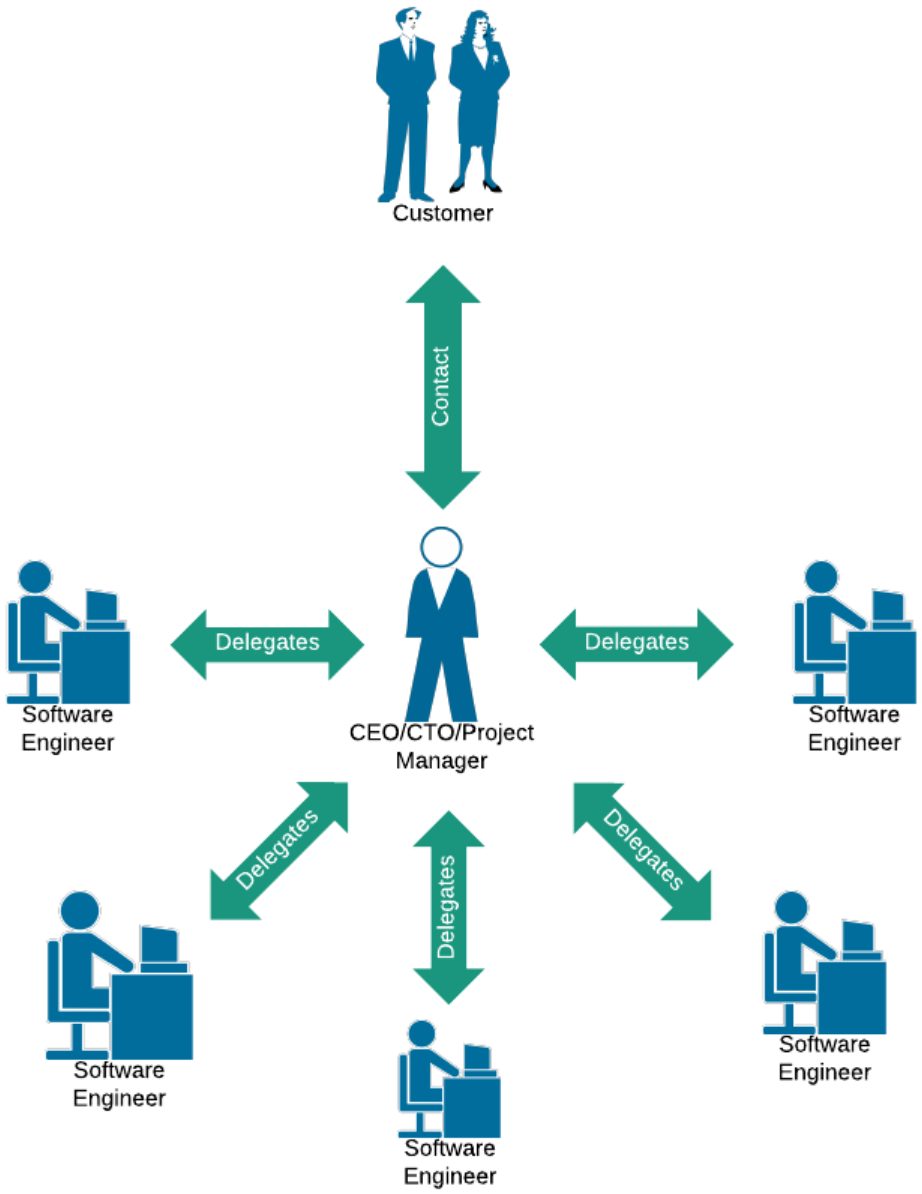

Figure 2.4: Team structure in the beginning. Different sizes of software engineers related to different amount of working-hours per week.

In the middle of the figure, the founder is positioned. He only delegated development tasks to part-time software engineers. So there were a lot of responsibilities left:

- Executive responsibility in his role as CEO
  - customer acquisition
  - human resources
- Technical responsibility in his role as CTO
  - defining and maintaining a software engineering process
  - keeping track of the technical portfolio
  - keeping track of code quality
  - maintaining technical infrastructure
- Finances in his role as CFO
- Realisation of projects in his role as the project manager
  - Assign software engineers to projects and tasks and take care of resources
  - Customer support
  - Creating offers including requirement engineering and time estimation
  - Set up milestones and deadlines

These responsibilities are far too much. In the beginning, this amount of responsibilities was only manageable, because the number of employees was lower. The approach to handle this number of responsibilities was to employ a project manager because this responsibility was the easiest to delegate. The integration of the project manager is shown in figure 2.5. The crucial step was handing over the customer after acquisition to the project manager. Therefore the project manager was involved in meetings with customers as early as possible to guarantee a smooth transition.

The concrete employed project manager was again a student with part-time employment. In the end, this solution had not the effect of taking away all the project management work of the founder, but at least the workload got less, and the so-called truck factor was dramatically decreased by the introduced second person, who was able to run the daily business and give feedback to customer requests. This situation is the starting point of this master thesis.

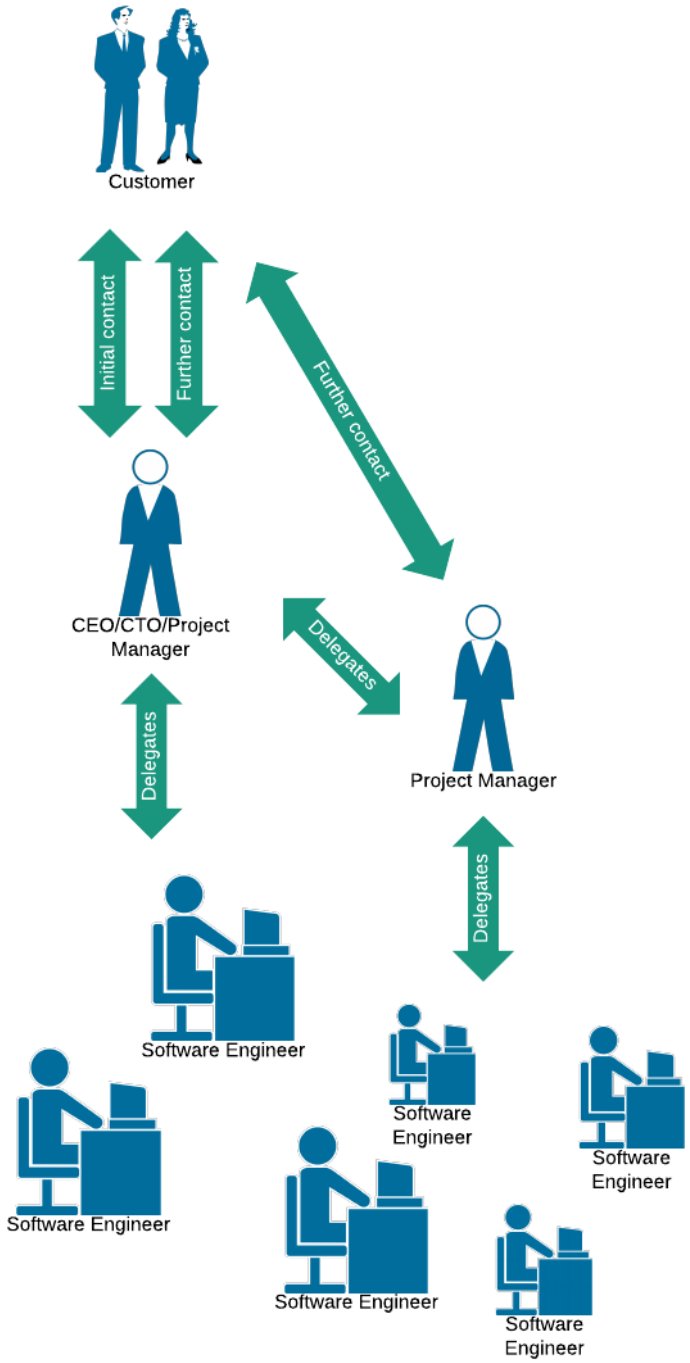

Figure 2.5: Current team structure. Different sizes of software engineers related to different amount of working-hours per week.

2.1.3 Drawbacks

Now that the situation of the company is described comprehensively, the drawbacks of this situation need to be discussed. These drawbacks can be derived from the list of responsibilities of the founder by categorizing the list entries in urgent and not urgent. It has to be mentioned that all of those entries are important. Therefore an additional segmenting is not necessary. The distinction between urgent and not urgent is therefore interesting because urgent tasks are processed before the not urgent ones. As a result, the urgent tasks are delivered with more quality and integrity than not urgent tasks. Not urgent tasks are therefore often neglected, unattended or even omitted. In the following the same list of responsibilities is provided again, whereby the urgent tasks are presented boldly.

- **Executive responsibility in his role as CEO**
  - **customer acquisition**
  - **human resources**
- Technical responsibility in his role as CTO
  - defining and maintaining a software engineering process
  - keeping track of the technical portfolio
  - keeping track of code quality
  - **maintaining technical infrastructure**
- **Finances in his role as CFO**
- Realisation of projects in his role as the project manager
  - **Assign software engineers to projects and tasks and take care of resources**
  - **Customer support**
  - Creating offers including requirement engineering and time estimation
  - Set up milestones and deadlines

In the following, the consequences of neglecting those responsibilities will be discussed in more detail.

No defined software engineering process

One of the reasons for the not fully successful implementation of a project manager is the lack of a structured process. A structured and defined process also defines the roles and their responsibilities. This clear separation of responsibilities makes it easier to integrate new employees into the running system. A chaotic and running system could be effective if each member knows its role and its responsibilities, but this takes a long time. And if it is easier to integrate new employees, it is easier to scale without omitting important responsibilities.

Not keeping track of code quality and technical portfolio

Working software is not necessarily a good software. A lack of code quality and a bad technical portfolio are causing high costs in the lifecycle of a software. Especially a lack of code quality is introduced when tasks are urgent, and no senior developer is monitoring the quality of the integrated code. This will be discussed in high detail in section 3.1.7.

Poor requirement engineering and time estimations

A poor requirement engineering leads to poor time estimations. Most often poor time estimations appear as an underestimated amount of work needed, because some critical points may be missed. A contract with an underestimated amount of work is therefore often combined with unrealistic deadlines. Those deadlines put developers under pressure and support the process of building technical debt. In addition to that, poor requirement documents do not support the common understanding of tasks sufficiently. This could lead to implemented solutions, which do not meet the customer's expectations. In this case, the customer is often in a better position, and an adjustment or a reimplementation is needed. This leads to higher costs and an increasing probability of missing deadlines. In extreme cases, this could cause a project to be canceled and therefore a significant loss of money.

2.2 Elaboration of Approach to enable Project Success

As shown in the previous section, there are many problems in the company to investigate. The overall goal is to enable project success. The company's success itself would exceed the scope of this work. In order to find an appropriate approach to achieve the goal of project success, a definition of the terms project, project management, and project success has to be provided (2.2.1).

In the subsequent section (2.2.2) the success factors mentioned in the literature are discussed. One of these factors is the process used. The process, or rather software process improvement, will be the starting point on finding an approach 2.2.3. It will also be shown, that software process improvement is closely related to software quality management. This starting point serves as the basis for the concrete approach chosen in section 2.3.

2.2.1 Project, Project Management and Project Success

To write a master thesis about enabling project success, the term project and the meaning of project success need to be defined. An early definition of project management by Richard Paul Olsen [65] helps by identifying, what a project is:

> Project Management is the application of a collection of tools and techniques (such as the CPM and matrix organisation) to direct the use of diverse resources toward the accomplishment of a unique, complex, one-time task within time, cost and quality constraints. Each task requires a particular mix of these tools and techniques structured to fit the task environment and life cycle (from conception to completion) of the task.

From that definition, the definition of a project as a unique, complex and one-time task, can be derived. In addition to that, it also defines the goals of a project, namely meeting time, cost and quality constraints. Reaching or not reaching these goals can serve as an indicator if a project succeeded or failed. Roger Atkinson states that this definition tries to reference the views of the 1950's also known as the Iron Triangle (see figure 2.6), whose origin is not clear [4].

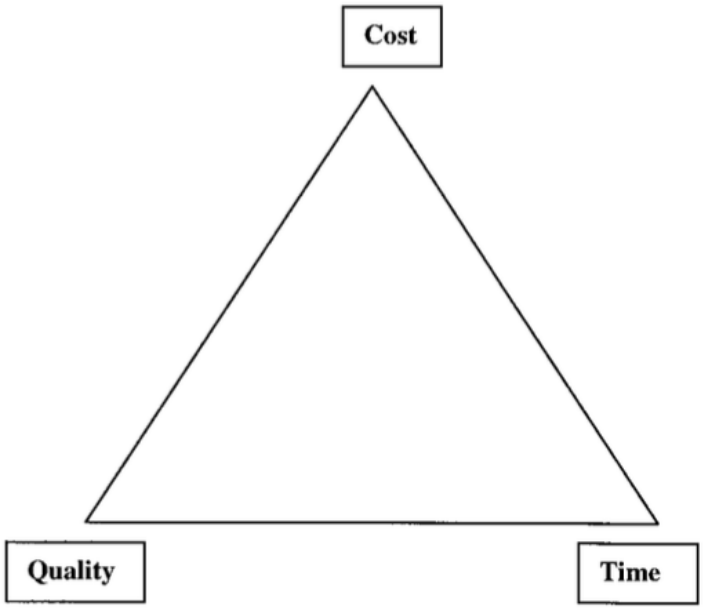

Figure 2.6: The Iron Triangle [4]

Another definition of project success including interrelations between the factors provides Harry Sneed in his book about Software Management [81]. He is describing four dimensions:

1. Quality
2. Scope
3. Timeliness
4. Cost

The interrelations described states, by changing one of the factors at least one of the other factors will change (intended or not intended), e.g., if scope gets bigger, more budget or time is needed or even both. If these additional resources are not provided, the quality will suffer.

For years meeting time, quality, cost, and scope goals served as the basis for judging the project. Therefore a project is stated to be done right, if

- it was delivered in time,
- if it meets all requirements by the customer,
- if it was developed within estimated cost and effort and
- if some predefined quality parameters are met.

Roger Atkinson notes that such a project is not necessarily really done right, if for example the project is not liked by the sponsors, not used by the customers or did not improve the effectiveness or efficiency of the organization [4]. Therefore only considering time, cost, quality, and scope does not suffice, to state, that a project was successful or not. This originates in the fact, that projects are most often assessed right after their delivery. This limits the criteria for determining project success to the Iron Triangle and excludes possible longer-term benefits of a project. Atkinson suggests the four dimensions for understanding project success, called the square route (see table 2.1). In it, the Iron Triangle is only one dimension and is extended by the information system, organizational benefits, and stakeholder benefits.

Paul Bannerman also investigated in finding a definition for project success, which expands the Iron Triangle [9]. He proposes five levels of project success:

1. Process success (process selection, alignment, and integration)
2. Project management success (time, cost, scope)

Table 2.1: Square route for understanding success criteria [4]

Iron Triangle	Information System	Organizational Benefits	Stakeholder Benefits
Cost	Maintainability	Improved efficiency	Satisfied users
Quality	Reliability	Improved effectiveness	Social and
Time	Validity	Increased profits	Environmental impact
	Information	Strategic goals	Personal development
	quality use	Organizational-learning	Professional learning
		Reduced waste	Contractors profits
			Capital suppliers
			Content project team
			Economic impact to
			surrounding community

3. Product success (quality, requirements, effectiveness, acceptance, use, satisfaction, impact)
4. Business success (business plan, meeting expected benefits for the company)
5. Strategic success (market, industry, competitive- investor, regulator and other impacts

2.2.2 Enabling Project Success

In order to be successful, the requirement is to meet the critical success factors (CSFs). Christine V. Bullen defines CSFs as follows [13]:

> CSFs are the limited number of areas in which satisfactory results will ensure successful competitive performance for the individual, department or organization. CSFs are the few key areas where ”things must go right” for the business to flourish and for the manager’s goals to be attained.

Chow presented a study based on those CSFs [20]. In the study, the CSFs are investigated in terms of agile software development project success (quality, scope, time and cost). First of all, 48 factors were collected and categorized. Subsequently, those factors were analyzed to find out, which of those are critical. The factors were classified in the following categories:

1. Organizational factors
2. People factors
3. Process factors
4. Technical factors
5. Project factors

These relations including examples of different categories are also illustrated in figure 2.7.

The first thing to distinguish is if factors are controllable or not. In the company under investigation three categories are not changeable easily: Organizational factors, people factors, and project factors. Process and technical factors are remaining.

These arguments directly point to the area of software process improvement, as process factors relate to one out of two possible success factor categories. In addition to that, many technical factors can be incorporated and are therefore considered in the field of software process improvement. The next section will present the field of software process improvement, including its aim and the connection to the overall goal of this master thesis, namely project success.

2.2.3 Software Process Improvement

The previous section derived SPI as the field of interest to enable project success. To check, whether SPI is a good starting point to enable project success, the thesis is double checked by examining the reasons for doing SPI. Schmitt and Diebold [77] found four common improvement goals by looking at existing literature combined with an additional survey and a set of workshops. The goals are listed subsequently and are shown in more detail in figure 2.8:

1. Customer involvement
2. Organizational democratization
3. Quality
4. Time-to-market

Whereas quality and time-to-market directly correspond to components of project success (see 2.2.1).

As quality is one of the improvement goals, it seems, that quality management is

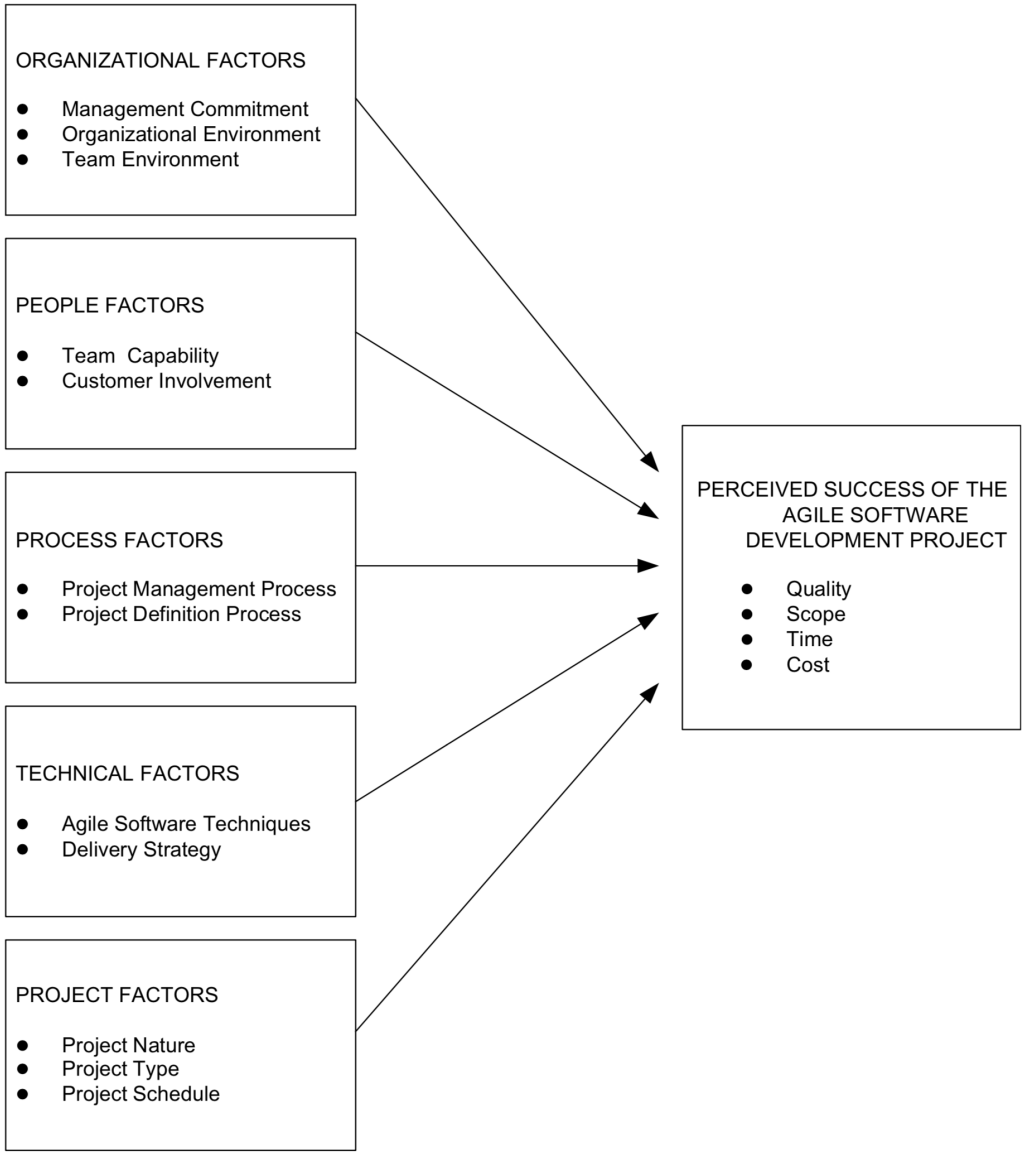

Figure 2.7: Categorization of CSFs [20]

Figure 2.8: Improvement goals [77]

strongly related to SPI. This is confirmed by Jacobsen et al. [42].

Also, the trends of SPI collected by Kuhrmann et al. show that SPI is the right choice even for SMEs [47]. These SPI trends are enumerated in the following:

1. New and customized SPI models
2. SPI success factors
3. SPI for SMEs
4. SPI and agility

Finally, the concrete approach used, to improve the software process in the company under investigation is elaborated and described in the next section.

2.3 Approach

In the previous section, SPI is chosen to enable project success. SPI was chosen because process factors are one of the critical success factors of projects. In addition to that, the reverse direction was investigated. It turned out that the aim of SPI supports two out of four success criteria of the iron triangle (see 2.2.1).

In this section, an appropriate SPI approach for very small web companies is chosen. On the one hand, this is done by showing an overview of SPI frameworks, which are characterized by different properties and on the other hand by looking at the special needs of very small companies.

2.3.1 SPI Frameworks and Properties

Process improvement can, at its core, be described as consistently applying the practices that give one good results and changing the practices that cause problems [92]. The overall assumption of process improvement is that improving the quality of the process will improve the quality of the product. Many methods, models and techniques emerged over the years. It is not necessarily about the actual practice itself [44]:

> [...] the success of a project depends heavily on the implementation maturity, regardless of the process model.

SPI framework is the term used to put them all together.

In the following four different SPI frameworks are presented, namely:

- ISO 9001
- CMMI - Capability Maturity Model Integration
- ISO 15504 (SPICE)
- QIP/GQM - Quality Improvement Plan combined with the Goal Question Metric approach

The first three frameworks are selected mainly by their popularity. Whereas QIP/GQM is not that popular anymore, but it is also included in the list because its nature is very different compared to the other frameworks.

The following presentation of the frameworks consists of basic descriptions including the structures and the detailed investigation of the following properties for each framework:

- Improvement initiation
- Progression
- Process improvement method
- Assessment method and authority

These properties will help to compare the frameworks. They are a subset of the properties used in the taxonomy for SPI frameworks elaborated by Halvorsen and Conradi [33]. Only those properties were described, which should guide the selection of the framework used within this action study.

Properties

Before presenting the frameworks, the selected properties are explained in detail.

Improvement initiation is about the viewpoint of process improvement, whether the improvement task is approached top-down or bottom-up. Thomas and McGarry wrote an article about it, whereas McGarry states, that process improvement is a bottom-up task, and Thomas, that it is a top-down task [87].

Thomas argues that most of the companies do not even have anything, which can be recognized as a software process. Moreover, as in any other engineering discipline, the requirements have to be settled. He describes a gap analysis (comparison between current practices and a standard) as a valuable place to start process improvement. This relates to a top-down approach.

The main argument of McGarry is that the assumption that an improved process will improve the product is not proven. Also, more important, ”if a changed process has no positive effect on the product, there is absolutely no value in making it”.

In other words, the top-down approach tries to close the gap between a company and a generalized standard and the bottom-up approach focuses on specific goals for each company individually.

Progression is about the way improvement is achieved. There are three different possibilities (also shown in figure 2.9):

- Flat
- Staged
- Continuous

ISO 9001 is a typical flat framework. A company is certified or not. There is no possibility of being more or less certified. Whereas CMMI is a staged framework. There exist different maturity levels, which show the quality of the process. In the end, QIP/GQM uses a continuous representation. Its focus is only on the evolution of the company.

The type of progression is a crucial factor in terms of comparability. A flat framework can distinguish only certified and not certified. Whereas a continuous representation has, of course, the highest level of detail, it is hard to compare. For comparison, a staged environment is the best.

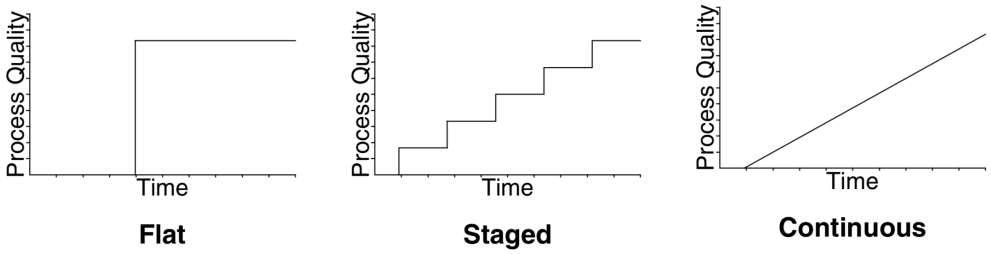

Figure 2.9: Types of progression

Process improvement method ”defines the strategy and the process by which improvement is pursued” [17]. There are also SPI frameworks existing, which are purely assessment frameworks without any guidance to improve the process.

Popular process improvement methods are:

- PDCA - Plan, Do, Check, Act
- IDEAL - Initiating, Diagnosing, Establishing, Acting, Learning
- QIP - Quality Improvement Plan

PDCA also called Deming wheel is an iterative four-step management tool to improve processes. At first, the actual processes are assessed and actions are planned

(Planning). Subsequently, those actions are executed (Do). As the third step, the results are checked against the expectations (Check). Whether these expectations are met, the trial process is accepted as the new standard and as a new base for the next planning step (Act).

IDEAL is shown in figure 2.10. The five steps are described as follows [15]:

1. Initiating - Laying the groundwork
2. Diagnosing - Determining where one is and determining future goals
3. Establishing - Planning how one is going to achieve goals
4. Acting - Doing the work required to reach the defined goals
5. Leveraging - Learning from what has been done for the next iteration of the process improvement cycle

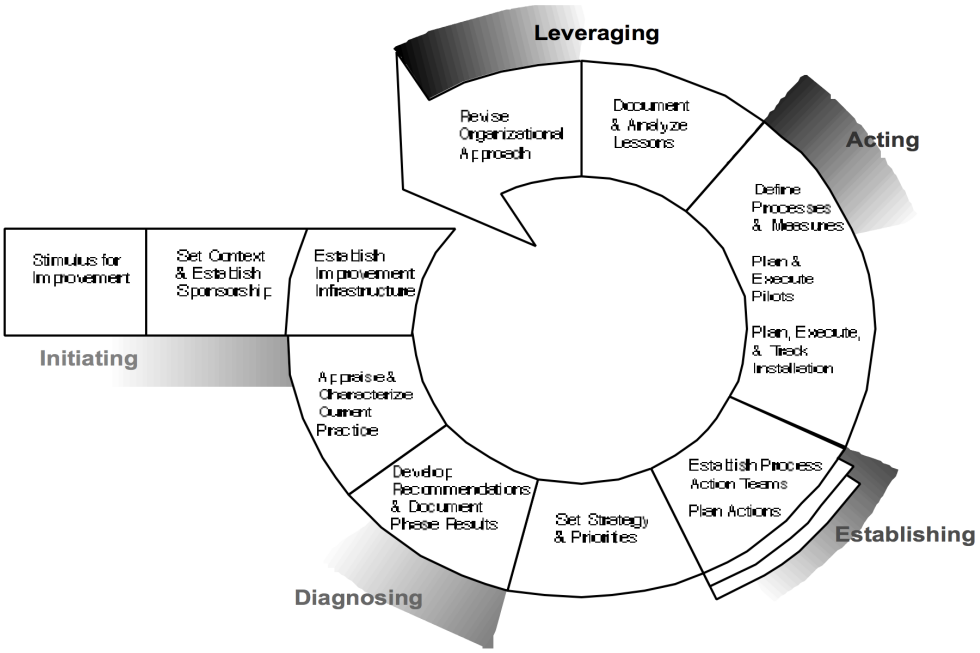

Figure 2.10: The IDEAL model [56]

QIP is the last process improvement method to be explained. It is based on the PDCA approach. It is a quality approach that aims to learn from experience. It

is built upon experimentation and application of measurement. The improvement loop can be summarized as [82]:

> Each new development project is regarded as an experiment and available results of every foregoing and ongoing experiments should be packaged and reused, too.

The cycle consists of the following six steps [82]:

1. Characterize - Understand the environment and existing processes
2. Set goals - Set quantifiable goals
3. Choose process - Choose appropriate processes to reach the goals
4. Execute - Execute the elaborated processes
5. Analyze - Evaluate the current practices at the end of each specific project including recommendations for future projects
6. Package - Combine the experience gained and update the experience base

Assessment method and authority describe in which way the maturity or the conformity is determined. Some of the frameworks include a certification, of course only issued by an external authority. Others can be used internally and externally. At last, others do not even provide any assessment at all.

Frameworks

In this section, the different SPI frameworks are presented.

ISO 9001 defines the requirements to a quality management system QMS [84]. The newest version is ISO 9001:2015 and replaced the previous version ISO 9001:2008. The International Organization for Standardization ISO defines a QMS as ”a way of defining how an organization can meet the requirements of its customers and other stakeholders affected by its work”. It is based on continual improvement.

ISO 9001 requires organizations to define objectives relating to quality and meeting customer needs themselves and to improve their processes continually in order to reach them. ISO claims that ISO 9001 is suitable for organizations of all types,

sizes, and sectors. The standard is based on seven principles [85] defined in ISO 9000, which contains a detailed explanation of these principles and many of the terms and definitions used in ISO 9001. These principles ”are a set of fundamental beliefs, norms, rules and values that are accepted as true and can be used as basis for quality management”. The seven quality management principles are

1. **Customer focus**
   The primary focus of quality management is to meet customer requirements.
2. **Leadership**
   Leaders at all levels establish unity of purpose and direction and create conditions in which people are engaged in achieving the organizations quality objectives.
3. **Engagement of people**
   Engaged people at all levels of a company are essential to create and deliver value.
4. **Process approach**
   Results are more consistent and predictable when managed interrelated processes exist, which are part of a coherent system.
5. **Improvement**
   Ongoing focus on improvement is essential.
6. **Evidence-based decision making**
   Decisions based on analysis and evaluation of data will more likely produce the desired results.
7. **Relationship management**
   Organizations have to manage relationships with interested parties.

ISO 9001 is a top-down approach, whereby the top-level requirements are described in the standard.

An external auditor can certify a company. There is no continuous or staged assessment. Therefore ISO 9001 is a flat framework.

The standard itself does not define what has to be done by the company, but continuous improvement is one of the principles of the stand. Therefore a process improvement method needs to be established, but the standard does not explicitly determine one. PDCA as an example was the method to use in the previous version of 2008. In 2015 as mentioned, the regulations were relaxed, and the company solely makes the selection for a method.

CMMI as the abbreviation for ”Capability and Maturity Model Integration” is the successor of CMM (Capability and Maturity Model). Historically the first version of CMM was released 1991. 2002 CMMI was released. The most used version at the moment is CMMI 1.3, which was released in 2010. 2018 CMMI 2.0 was released. This discussion covers CMMI 1.3 as it is widely used and freely available. These are important points when comparing to ISO 15504. ISO 15504 was released later and is not freely available, which raises points of criticism. These points are discussed, when presenting ISO 15504 in detail.

The basic concepts of the CMMI are process areas, capability and maturity levels. There is a predefined set of process areas. The capability level describes the degree of institutionalization of a single process. These levels are:

0. Incomplete
1. Performed
2. Managed
3. Defined

The maturity level describes the maturity of a whole organization. These levels are:

1. Initial (ad-hoc and chaotic)
2. Managed (focus on basic project management)
3. Defined (focus on process standardization)
4. Quantitatively managed (focus on quantitative management)
5. Optimizing (focus on continuous process improvement)

To reach a certain maturity level, a defined subset of process areas have to have a certain capability level. For maturity level 2 all corresponding process areas have to have a capability level of 2. From maturity level 3 to 5 the corresponding process areas have to have capability level 3. All requirements of the lower level need to be fulfilled to reach a higher maturity level. This relationship is also shown in figure 2.11. The framework is therefore staged, and of course, it is a top-down approach. It can be assessed internally and externally by an auditor.

Like ISO 9001, CMMI does not define any specific process improvement method, but SEI, the founders of the CMMI, developed one, which can be used for implementing standards like CMMI and ISO 15504. It is the already described IDEAL model.

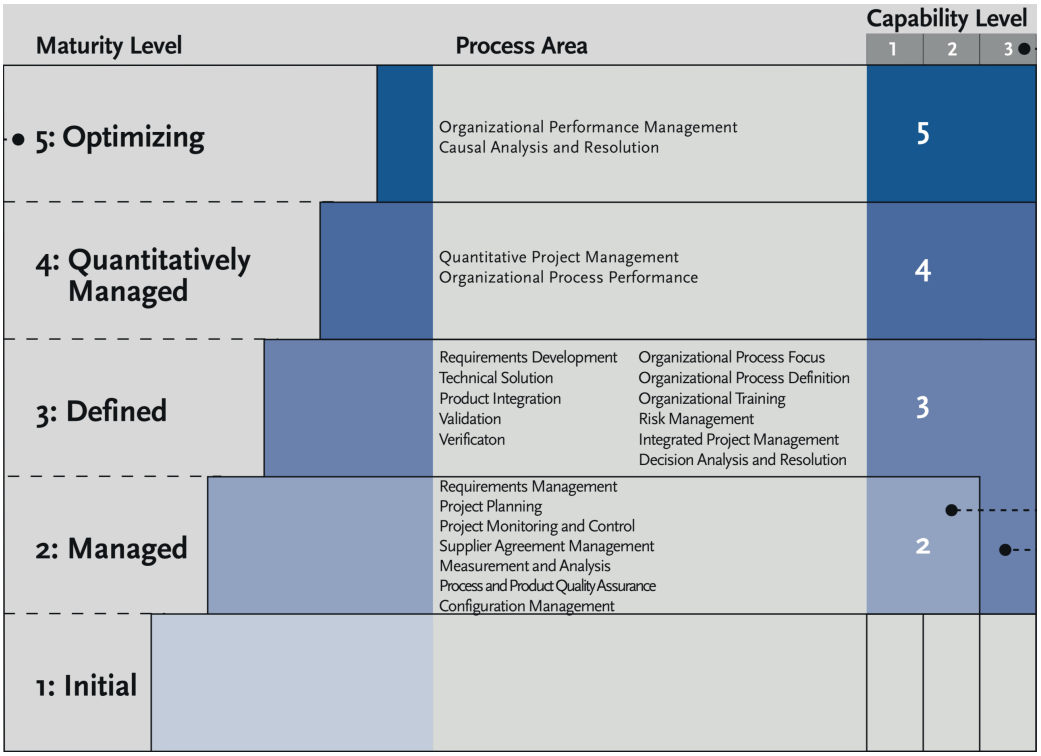

Figure 2.11: CMMI v1.3 [32]

ISO 15504 - SPICE is again a standard by the International Organization for Standardization. SPICE is the abbreviation for ”Software Process Improvement and Capability Determination”. SPICE is a standard containing different parts:

1. Concepts and vocabulary
2. Performing an assessment
3. Guidance on performing an assessment
4. Guidance on use for process improvement and process capability determination
5. An exemplary software life cycle process assessment model
6. An exemplary system life cycle process assessment model
7. Assessment of organizational maturity
8. An exemplary process assessment model for IT service management
9. Target process profiles
10. Safety extension

The concept of SPICE correlates with the concept of CMMI. It is about assessing process areas and the maturity of organizations. Part 2 and 3 cover the assessment of process areas, whereas part 2 is the normative part and part 3 gives guidance to fulfill the requirements of part 2. Part 7 uses the information gathered by the process assessment to determine an organization’s maturity level. The process areas themselves are not defined in ISO 15504, but they can be taken from ISO 12207 or ISO 15288. ISO 15288 defines general system lifecycle processes, whereas ISO 12207 focuses on software and defines software lifecycle processes. This relationship is shown in figure 2.12. The figure also shows the impact of ISO 9001. It assists by providing the quality management system requirements.

ISO 15504 in combination with ISO 12207 is therefore only slightly different to CMMI. As already mentioned in the section about CMMI, this raises criticism. On the one hand, the meaningfulness has to be scrutinized because it is developed years after the release of CMM. On the other hand, it is not that widespread, because it is, in contrast to CMMI, not freely available.

ISO 15504 is of course, as CMMI, a staged top-down approach. It can be assessed by an official auditor, which has to fulfill the requirements of part 2. IDEAL can be used as a process improvement method. It is also conceivable to use PDCA, as it was mandatory in ISO 9001:2008 and ISO 9001 provide the requirements of a quality management system to ISO 15504.

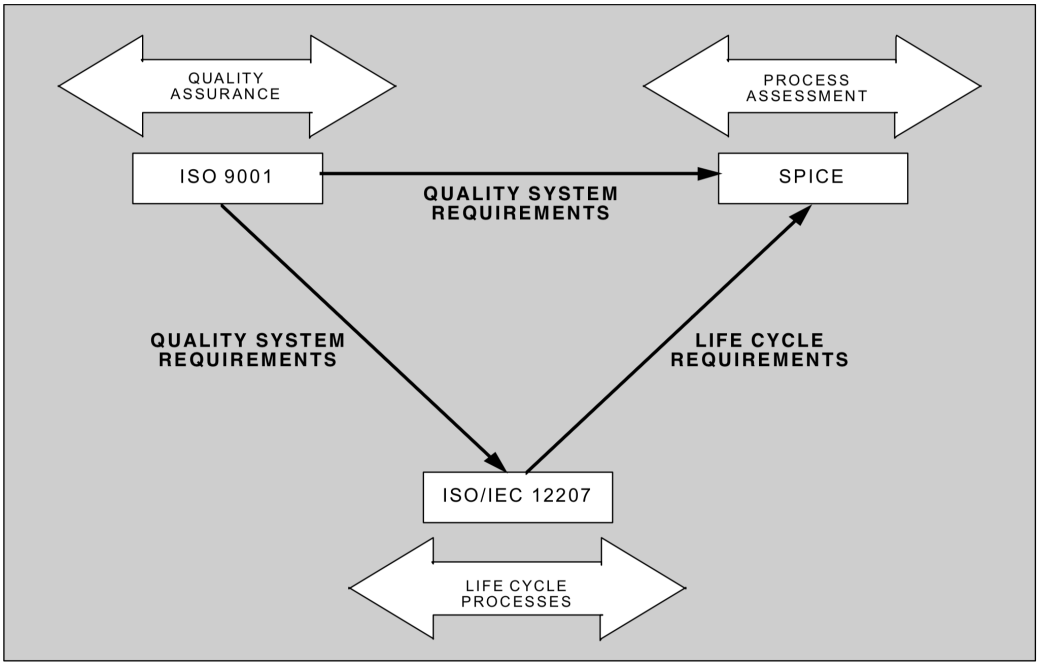

Figure 2.12: Relationship of ISO 15503, ISO 12207 and ISO 9001 [32]

There are also several attempts to harmonize the standards described so far, e.g., ISO 15504 to CMMI [67][73]. The CMMI Institute itself compared ISO 9001 and CMMI [40]:

> In particular, the synergy between ISO 9001 and CMMI is high. But the emphasis in CMMI is on assuring institutionalization - across multiple projects - that represent the organization being appraised. As a result, we sometimes find that a ”Maturity Level 3” organization easily passes its ISO auditor’s look, but that ISO organizations are sometimes not of ”higher maturity.”

There are also attempts for concrete mappings [7][94][66].

QIP/GQM - Quality Improvement Plan combined with the Goal Question Metric approach is very different from the frameworks described so far. This approach is bottom-up. The idea is to learn from project to project. The approach ”uses internal assessments against the organization’s own goals” [44]. Solingen and Berghout [82] describe this combination of techniques and practices. For these assessments, an analysis method is needed.

The Goal/Question/Metric (GQM) approach is used for this purpose. Basili et al. [14] introduced the GQM model. It is a top-down approach to find the right metrics to determine, whether a goal is reached or not. For that purpose one sets out one or more goals. Then questions, which need to be asked to determine whether a goal is achieved or not, are elaborated. Concrete metrics then answer those questions. This is also shown in figure 2.13.

This combined approach is therefore bottom-up and has a continuous progression. The assessment is internally, and the process improvement method is the QIP approach.

2.3.2 Situation in Very Small Enterprises

This section will show the relevance of SPI for SMEs and if it is a common practice. Also, different solutions will be presented, and finally it is explained, why SPI

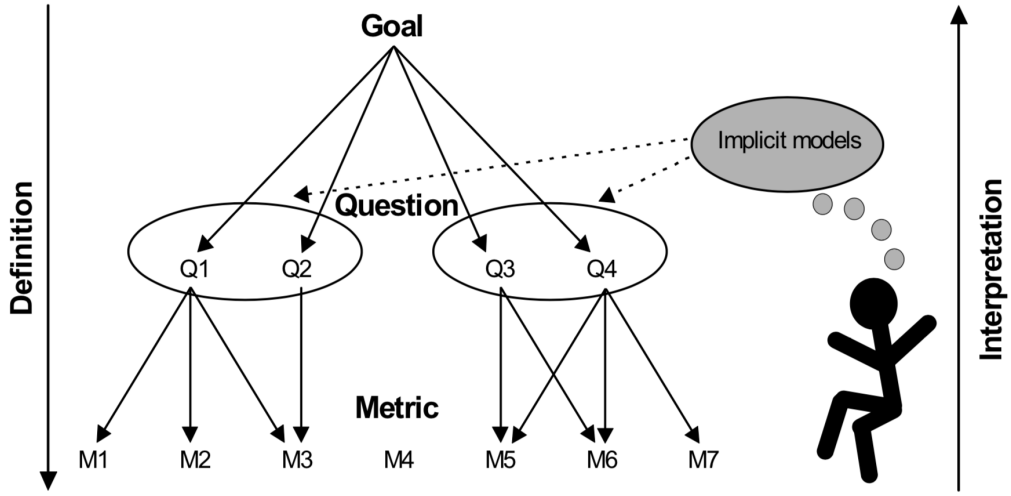

Figure 2.13: GQM Approach [82]

initiatives are often not used in SMEs.

At first, SMEs have different needs compared to large companies, because they are not just scaled-down versions of those [86]. This is relevant because many of the SPI frameworks presented were developed for large companies. Richardson and Von Wangenheim noted that small companies are ”extremely responsive and flexible because that’s their advertised competitive advantage” [41]. They also stated, that, in contrast to large companies, small companies do not have enough staff to develop functional specialties and also have tight finances as an additional constraint. Therefore the SPI frameworks are often not suited for small companies, especially established standards.

As SMEs are a big fraction in terms of economy, this topic is for interest. Therefore Kuhrmann et al. have found in their systematic literature review, that SPI in SMEs and customized approaches are trends in current research [47].

Nevertheless, SPI is not useless for small companies. SMEs reported short- and long-term benefits from SPI initiatives [74][75][16][30][60][12][29]. However, studies also found that small companies sometimes cannot afford the costs of implementing comprehensive frameworks [38][64]. As a result, the research

tried to evaluate which process areas are of high importance to small companies [53][62][61][93]. Most of those studies do not try to scale-down a framework. Instead, they try to find a subset of process areas, which better fits small companies. Mc Caffery et al. presents one of those approaches. They propose the following six process areas [55]:

- Project Monitoring and Control
- Project Planning
- Requirements Management
- Configuration Management
- Process and Product Quality Assurance
- Measurement and Analysis

2.3.3 Selection

As shown in the previous section, the topic of SPI in small companies is under active research, and a final answer is not found yet. As most of the research is focused on top-down approaches, which are using a subset of process areas of high standards, the QIP/GQM bottom-up approach is selected for this thesis. In addition to that, the cost factor of even small-sized standards would exceed the budget of the investigated company. Also, as there is nearly no structure in the company yet (see section 2.1), a bottom-up approach, which is not introducing a big change at once seems to be more appropriate. This iterative character also helps in financing the change. The idea is to use a small proportion of a project's budget for improvement.

The concrete improvement process is shown in figure 2.14. The scope of this work includes two iterations of the QIP cycle. The concrete implementation and the results can be found in section 3.

The concrete steps, which can also be found as the structure of each iteration in this work are described in the following. At first, a project is observed, and problems tried to be identified by reflecting, what happened during the project (characterization step of QIP). By inverting the major problems, the goals are set (the problem should not occur again). This represents the step of setting goals. These goals are used to initialize or extend the GQM model. Before executing the third step of the QIP (choose process), the observed problems of the preceding

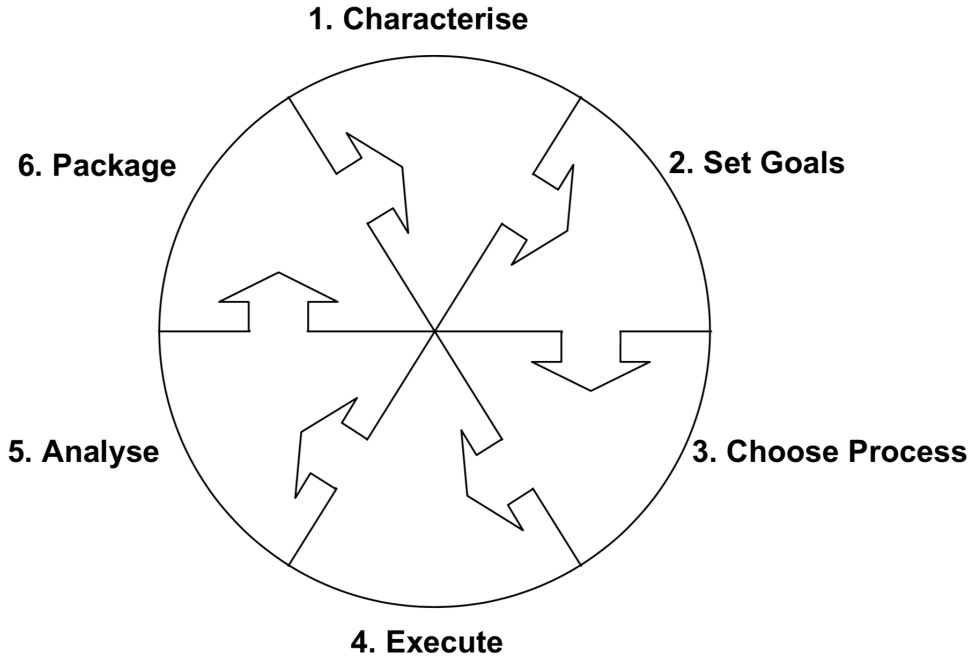

Figure 2.14: Quality Improvement Plan [82]

project are quantified. In QIP this would be done in the analyzations step, but it seems to be a better idea to validate the problems observed by looking at the concrete measures. This could save time and money if the quantification does not overlap with the observations. The following steps of choosing corrective actions (QIP: choose process) and analyzing the impact of those (QIP: Analyze) are done as proposed. The last step of packaging ist neglected.

3 Implementation

Chapter 2 describes the used methodology of this work. In this chapter, two concrete iterations are presented. Following the methodology, each iteration affects two projects. A legacy project is used to reflect the current state. The corrective actions are applied to a subsequent project, which is then used to validate the impact of those actions. In the first iteration, project A is the base project, and project B is the project on which the corrective actions were applied. Project B is then used as the basis for the second iteration. The corrective actions of this iteration are then applied to Project C and of course validated by the elaborated quality model. The structure of the text follows the six steps of the used methodology. In addition to that, descriptions of the projects A and B are provided in the section of the first iteration and a description of project C is provided in the section of the second iteration. These descriptions are important for the subsequent interpretation of the results. It is important to mention that the author of this work managed all three projects. Of course, this was beneficial for this research because it allowed the author to control and to observe the circumstances of the projects better than by only providing consultancy.

3.1 Iteration 1

Each iteration consists of six steps, whereas each of those has its subsection. In addition to that, project A and project B are described in two additional subsections.

3.1.1 Description of Project A

The goal of project A is the development of a web application. The web application should help to schedule courses automatically. In addition to that, it should be possible to manage instructors and participants. PHP was used as backend in combination with the content management system Concrete 5. The frontend was built by using client-side JavaScript and partially server-side PHP. The initial team consisted of two inexperienced developers, who were students working full-time in the summer break. They had almost no experience with PHP and no experience with Concrete 5 or JavaScript. As already mentioned, the project was managed by the author of this work. By the end of the summer break, the team composition changed. The two actual developers left the team, and the project manager started to contribute also to the code, approximately at that time, when the big change of requirements occurred. The project manager had already one year of part-time experience with frontend JavaScript. He was supported by a colleague, which was already familiar to some extent with the Concrete 5 PHP framework used in project A.

3.1.2 Observations of Project A

As mentioned already, the goal of project A was developing a web application, which serves as a tool to schedule courses automatically and to help managing instructors and participants. The specification was poor and written by the customer, which has little experience in creating specifications. The document was rather a wish list containing some vague requirements mixed with fixed solutions. This document was not reviewed properly either by the project manager nor the CEO. Nevertheless, it served as the specification and therefore as the acceptance criteria for fulfilling the contract and completing the project.

Observation A-1: Poor and vague specification signed off

The development team consisted of two inexperienced web developers. They were not familiar with the technologies used. The responsible person for the project was the project manager. The project manager was responsible for the product, the team and the communication with the customer. Therefore the project manager incrementally derived features from the specification. The features were delivered

continuously by the development team and continuously reviewed concerning functionality by the project manager. This functionality was regularly presented to the customer. The quintessence of the customer's feedback was that the product still does not fulfill all points of the specification. The project manager did not take that seriously, because he was conscious about not yet fulfilling all requirements, but was erroneously convinced, that the direction was correct. In the first testing phase of the customer's project manager, the project team got faced with an enormous amount of little adjustments in design and functionality. At this point, changes were already pretty expensive, because the project team already developed a high amount of technical debt. After integrating these adjustments, the application was presented to the end-users the first time. At this point, the project manager found out, that the end-users were not involved in creating the specifications and therefore the product did not solve any of the real problems the end-users were facing.

Observation A-2: End-users not involved in creating the specification

From that point on, the project was at risk for a long delay. Arguing, that these major changes were not part of the contract, was hardly possible, because:

- the customer was the biggest revenue generator for years,
- the specification was written vaguely and
- the deadline for delivering the product was already missed.

The result was that almost every change request was defined as a defect or at least as part of the contract. This is a dynamic, called scope creep, which typically occurs in small and medium enterprises [68]:

> Large software companies have dedicated teams whose only concern is software maintenance and evolution. VSSEs do not have that luxury; the development team is also responsible for handling incoming change requests.

The paper exactly describes also, how scope creep arises [68]:

> [...] customers often ask for changes while development has already started. Customers are often eager to have the project delivered and often sign off the requirements document without having done a proper analysis. When early builds of the software are to be delivered or demonstrated, the clients start to realize that the product being built it is not exactly what they want. Another factor is that new ideas emerge

> - unforeseen requirements - during development. These factors cause scope creep. [...] The scope of the project grows even more when unforeseen requirements are included.

Scope creep is also considered as one of the major risks in software projects [90], as the clients request changes but do not provide enough additional resources to finish the project on time [68].

Observation A-3: Scope creep

This led to many code changes in a stressful situation. With a high code quality, it is possible to cope with such a situation. However, the code base had a high amount of technical debt built in. In addition to that, no automatic test suite existed. Changing the code was therefore risky. Changing the code often was hardly possible without introducing a high number of defects. The customer often found these defects. The trust and patience of the customer continuously decreased.

Observation A-4: No regression tests

Observation A-5: Bad code quality

This happened, although the project manager invested much time in testing new functionality and features, which already worked. The costs exploded, and the project was not built on time, because, as described above, the customer neither signed off additional resources nor accepted an extension of the deadline.

Observation A-6: High manual testing effort

Also, the deployment itself was risky. Problems with the database version on different environments occurred often. Also, functionality which worked on development and staging environments often did not work in production.

Observation A-7: Deployment risky and error-prone

Observation A-8: Problems with configuration management

As already mentioned, a massive technical debt was built. This was not only caused by many changes, but also by the inexperience of the development team. The added code was never reviewed by at least one senior developer.

Table 3.1: Observations of project A

ID	Description
O-A-1	Poor and vague specification signed off
O-A-2	End-users not involved in creating the specification
O-A-3	Scope creep
O-A-4	No regression tests
O-A-5	Bad code quality
O-A-6	High manual testing effort
O-A-7	Deployment risky and error-prone
O-A-8	Problems with configuration management
O-A-9	Code contributions were not reviewed by a senior developer
O-A-10	Project was not delivered on time
O-A-11	Customer not satisfied
O-A-12	Planned costs exceeded
O-A-13	Many regression and production bugs
O-A-14	No defined process for change requests

Observation A-9: Code contributions were not reviewed by a senior developer

Observation A-10: Project was not delivered on time

Observation A-11: Customer not satisfied

Observation A-12: Planned costs exceeded

The conclusion of the project is that concerning our definition of project success the project failed in every dimension. In table 3.1 the observations are collected.

3.1.3 Define Goals

Regarding the observations stated in table 3.1, the goal of this section is to find concrete goals, which are the basis for the sections that follow. The observations of project A are a mixed list of causes and effects. The fact, that end-users not involved in creating the specification (O-A-2), led to a poor and vague specification (O-A-1). Scope creep (O-A-3) describes and compromises all the final effects to one term

(O-A-10: Project not in time, O-A-11: Customer not satisfied, O-A-12: Planned costs exceeded). In addition to that, scope creep puts a lot of time pressure on the whole team and has, therefore, most likely an impact on code quality (O-A-5). Figure 3.1 shows the dependencies of the different observations.

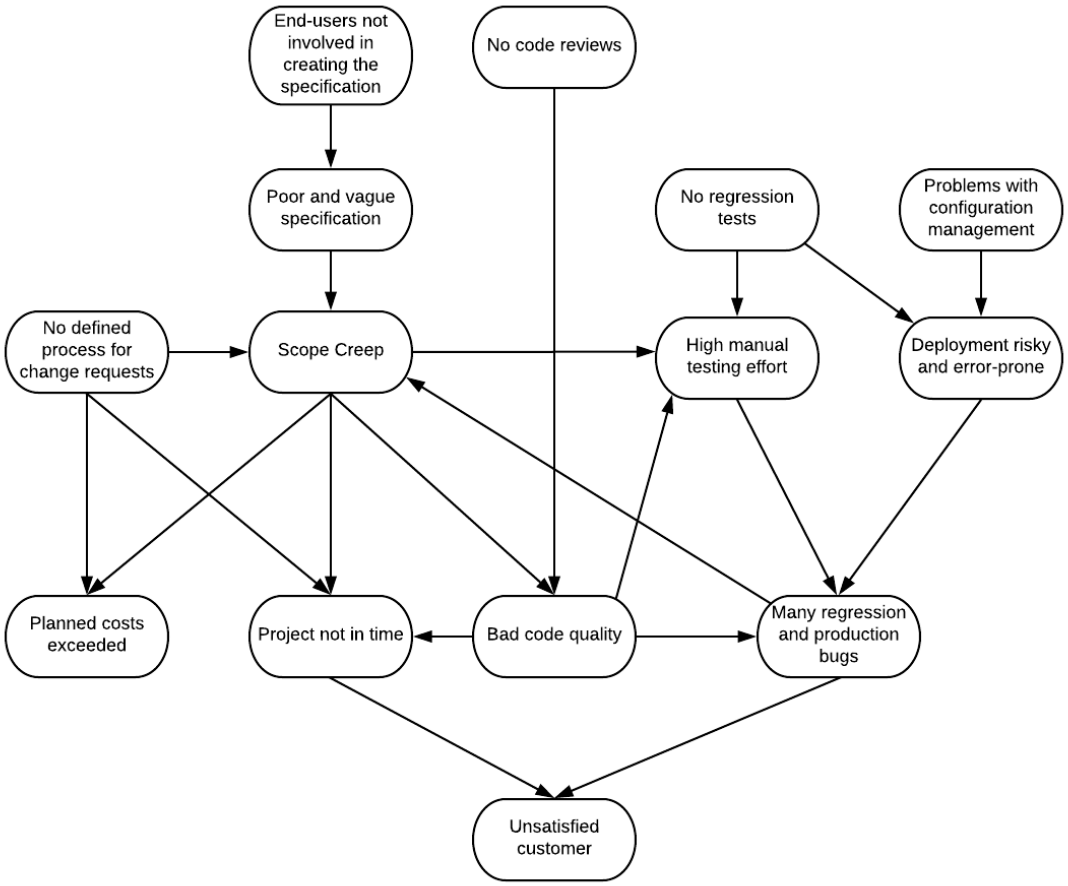

Figure 3.1: Dependencies of observations

This graph gives the possibility to identify the root causes, by merely enumerating all nodes without ingoing edges.

- Requirements development
  - Poor and vague specification signed off

  - End-users not involved in creating the specification
- No code reviews
- No regression tests
- Problems with configuration management
- No defined process for change requests

The goals of the initial GQM-model will be

1. preventing scope creep and
2. delivering maintainable code.

The questions and metrics for the GQM-model are elaborated in the following.

3.1.4 Initialize GQM-Model

The goal of this section is to derive a GQM-model from the observations stated in section 3.1.2. In section 3.1.3 the goals of preventing scope creep and ensuring a good code quality are listed and elaborated. In the following, the related questions and metrics are elaborated to initialize a full GQM-model. This GQM-model will be used to determine, whether the observations are consistent with the actual measurements and if the next project performed better.

Goal 1: Prevent Scope Creep

To determine if scope creep occurred or if the project was prevented from scope creep, the following questions need to be answered:

- Q1.1: Was the project delivered in-time?
- Q1.2: Was the project built within budget?
- Q1.3: How were the resources used?
- Q1.4: Was the feature estimation wrong?
- Q1.5: Were the change requests paid by the customer?
- Q1.6: Was the team successful in building trust by providing a reliable software in-time to put the project manager in a good change request negotiating position?

Metrics for Q1.1 - Was the project delivered in-time? To answer this question, the actual and desired project duration need to be taken into account. For the aim of comparability over different projects of different sizes, the metrics are given as ratios. The total numbers are also provided to give more contextual information [44]. The strategy of giving relative and absolute values will be used for all subsequent metrics.

The first two used metrics are therefore estimated project duration (M1) and schedule estimation accuracy (M2). The duration is considered as the period in calendar days including weekends from placing the order to the day of acceptance. The day of acceptance is defined as the day the bill was sent to the customer. The investigated company only sends the bill once the customer agrees with the delivered product.

$$\text{M1} = \text{Estimated project duration in calendar days}$$

Metric 1: Estimated Project Duration

$$\text{M2} = \frac{\text{Actual project duration in calendar days}}{\text{Estimated project duration in calendar days (M1)}}$$

Metric 2: Schedule Estimation Accuracy

Metrics for Q1.2 - Was the project built within budget? In contrast to question 1.1, which targets the in-time delivery, question 1.2 targets a development within the estimated effort. The total estimated project effort in hours (M3) and the effort estimation accuracy (M4) are used.

$$\text{M3} = \text{Estimated project effort in hours}$$

Metric 3: Estimated Project Effort

$$\text{M4} = \frac{\text{Actual project effort in hours (M5)}}{\text{Estimated project effort in hours (M3)}}$$

Metric 4: Effort Estimation Accuracy

Metrics for Q1.3 - How were resources used? If Q 1.1 and Q1.2 show that there are significant problems by staying in-time and within the estimated effort, the next question to be asked is, how the resources were used. This is done by looking at the effort distribution of the three different task types feature, failure and change request.

A feature describes functionality predetermined in the contract and the specification, whereas a change request defines a feature that was not part of the specification and should, therefore, be a reason to get a deadline suspension and additional budget. Another possibility is to substitute a predetermined feature with the change request.

The third type is a failure. In this thesis, the definition of IEEE Std 610.12-1990 [69] is used. The standard defines an error as:

> The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example a difference of 30 meters between a computed result and the correct result.

Fault (also called defect [44]) is the term for:

> An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program.

A failure relates to:

> An incorrect result. For example, a computed result of 12 when the correct result is 10.

Finally, a mistake is:

> A human action that produces an incorrect result. For example, an incorrect action on the part of a programmer or operator.

The term of interest, in this case, is failure. The reason for this is that each customer's complain about a failure causes effort and is documented. To determine if it is the same underlying defect would be object to a more detailed analysis, which is not needed in this case. For customer satisfaction, the failure rate is more important than the defect rate.

To provide comparability with other projects, the resource usage is given as proportion, whereas the actual project effort is used as the denominator and is also given (M5). These proportions are the feature effort proportion (M6), change request effort proportion (M7) and failure effort proportion (M8).

$$\text{M5} = \text{Actual project effort in hours}$$

Metric 5: Actual Project Effort

$$\text{M6} = \frac{\text{Effort for working on features in hours}}{\text{Actual project effort in hours (M5)}}$$

Metric 6: Feature Effort Proportion

$$\text{M7} = \frac{\text{Effort for working on change requests in hours}}{\text{Actual project effort in hours (M5)}}$$

Metric 7: Change Request Effort Proportion

$$\text{M8} = \frac{\text{Effort for working on failures in hours}}{\text{Actual project effort in hours (M5)}}$$

Metric 8: Failure Effort Proportion

Metrics for Q1.4 - Was the feature estimation wrong? In addition to the metrics of Q1.3, two metrics are provided to show if the project would have met the estimated effort, if there would have been no change requests (M9) and if there would have been no change requests and no reported failures (M10). Again the metrics are proportions. This time the estimated project effort (M3) is used as the denominator. If everything would be fine, if no change requests had occurred, change request management seems to work unsatisfactorily. Moreover, if also, in this case, the effort exceeds the estimated effort, two cases can be distinguished:

- The budget was too less to create such an application or

- the time used for working on failures was either not considered in the contract, or the quality was too bad, and the considered budget for working on failures was simply exceeded.

In other words, feature effort to estimation ratio (M10) shows the accuracy of the estimation, if there would have been worked only on predetermined features and no mistakes are made by the developers or at least no failures would have been reported.

$$\text{M9} = \frac{\text{Effort working on features and failures in hours}}{\text{Estimated project effort in hours (M3)}}$$

Metric 9: Feature and Failure Effort to Estimation Ratio

$$\text{M10} = \frac{\text{Effort working on features in hours}}{\text{Estimated project effort in hours (M3)}}$$

Metric 10: Feature Effort to Estimation Ratio

Metrics for Q1.5 - Were the change requests paid by the customer? The major negative effect of scope creep is an enormous amount of additional effort in the form of additional or changing functionality, which the customer is not willing to pay. A single metric is therefore sufficient, the billed change request effort proportion (M11). The metric shows, which proportion of the hours worked on change requests is billed.

$$\text{M11} = \frac{\text{Billed effort for change requests in hours}}{\text{Effort working on change requests in hours}}$$

Metric 11: Billed Change Request Effort Proportion

Metrics for Q1.6 - Was the team successful in building trust by providing reliable software in time to put the project manager in a good change request negotiating position? The reason for the non-existing willingness to pay for change requests is often caused by

- the situation that the customer is a critical revenue generator for the supplier and should therefore not be annoyed.
- a high number of failures, which puts the supplier into a bad position in change request negotiations.
- an already existing exceedance of the deadline.

The first reason is a fact and cannot be changed. Whereas the second and the third reason could be prevented. Therefore the customer found failure rate (M14) and the exceedance of the deadline (M2) are significant. The customer found failure rate is a relative metric. It shows the failures observed by the customer on average per month of a certain period[1]. The total values are also given. These are the total failures found by the customer (M13) in the period of observation (M12). The period of observation is defined by the first and the last appearance of a failure found by the customer.

$$\text{M12} = \text{Period of observation in days}$$

Metric 12: Period of Observation

$$\text{M13} = \text{Total failures found by customer in period of observation}$$

Metric 13: Total Failures Found

$$\text{M14} = \frac{\text{Total failures found by customer (M13)} * 30}{\text{Period of observation (M12)}}$$

Metric 14: Customer Found Failure Rate

[1] Typically this metric is given per license months [44]. As the reporting of failures is done by the customer's project manager, who is also responsible for the acceptance, the sum of failures per month of all users combined is more interesting in this case.

Goal 2: Deliver maintainable code

The following questions and metrics are necessary to determine if the goal of delivering maintainable code is achieved:

- Q2.1: How complex is the code?
- Q2.2: Is the code well documented by a meaningful test suite?
- Q2.3: How hard is it to change the code?
- Q2.4: Is the amount of technical debt acceptable?

There are many different ways to determine how difficult it is to understand, extend and change the code. Many of the metrics used to answer the questions above are derived from a free to use tool called SonarQube [2].

Metrics for Q2.1 - How complex is the code? The complexity of the whole system has a big impact if it comes to changing and understanding the code. Thus different metrics to measure complexity are provided. For these metrics, it is always interesting to see the average and the maximum, to detect outliers.

A first intuitive and common metric for complexity is the number of lines of code. By providing the lines of code, it is important to state, on what the metric is based on, because the operational definition of counting is ambiguous [44]. In the book ”Programming Productivity” different definitions are provided [43]:

- Count only executable lines.
- Count executable lines plus data definitions.
- Count executable lines, data definitions, and comments.
- Count executable lines, data definitions, comments and job control language.
- Count lines as physical lines on an input screen.
- Count lines as terminated by logical delimiters.

Jones [43] describes the result of the second method as ”logical lines of code”. This method is ”a somewhat more rational choice for quality data”[44]. In addition to the total logical lines of code (M15), the logical lines of code per file are considered, which can serve as an indicator if files are properly separated and no monoliths

[2] `https://www.sonarqube.org` last visited October 30, 2018

are created. This is done by providing the maximum logical lines of code per file (M16) and the average logical lines of code per file (M17).

$$\text{M15} = \text{Total logical lines of code}$$

Metric 15: Logical Lines of Code

$$\text{M16} = \text{Maximum logical lines of code per file}$$

Metric 16: Maximum Logical Lines of Code

$$\text{M17} = \frac{\text{Total logical lines of code (M15)}}{\text{Number of files}}$$

Metric 17: Average Logical Lines of Code

Cyclomatic complexity is a measure, that shows how many test cases are needed to test the whole control flow [54]. This measure ”was designed to indicate a program’s testability and understandability (maintainability)”[44]. The metric is based on graph theory and describes the ”number of linearly independent paths that comprise the program”[44]. As this metric is additive, the complexity of a file can also be calculated by summing up the cyclomatic complexity of each function block. Again the total cyclomatic complexity (M18), the maximum cyclomatic complexity per file (M19) and the average cyclomatic complexity per file (M20) are given.

$$\text{M18} = \text{Total cyclomatic complexity}$$

Metric 18: Cyclomatic Complexity

$$\text{M19} = \text{Maximum cyclomatic complexity per file}$$

Metric 19: Maximum Cyclomatic Complexity

$$\text{M20} = \frac{\text{Total cyclomatic complexity (M18)}}{\text{Number of files}}$$

Metric 20: Average Cyclomatic Complexity

A moderate to strong correlation between cyclomatic complexity and defect rate is found. As cyclomatic complexity is additive, it increases with the lines of code. Therefore it is still not clear, whether the correlation is caused by the increasing lines of code or the cyclomatic complexity itself. Studies, which tried to control the lines of code, are not consistent [44]. Initially, lines of code and cyclomatic complexity are measured. If it turns out, that they are having a strong correlation, one of those two can be neglected in future iterations.

Q2.2 - Is the code well documented by a meaningful test suite? A good test coverage supports developers in understanding and changing the code. As the first two projects had no test suite at all, question 2.2 is only determined by metric M21, which determines whether an appropriate automated test suite is applied or not.

$$\text{M21} = \text{Appropriate automated test suite applied}$$

Metric 21: Appropriate automated test suite applied

Metrics for Q2.3 - How hard is it to change the code? A high test coverage does not only improve the understandability of the code, but it also improves the robustness to change. It is easier to change because a well-designed test suite will immediately highlight if a change had broken existing functionality. Therefore test coverage (M21) is also used to answer this question.

The proportion of duplicated lines (M22) is also used to answer this question. Is this proportion high, the maintainability of the code decreases.

$$\text{M22} = \text{Percentage of Duplicated Lines}$$

Metric 22: Percentage of Duplicated Lines

Metrics for Q2.4 - Is the amount of technical debt acceptable? The cleanness of the code is determined by the proportion of duplicated lines (M22), which is already used in Q2.3. In addition to that, SonarQube provides an automatic inspection of the code, which identifies code smells and security vulnerabilities for example. They are all added up to the total number of issues and are broken down on issues per 1000 lines of code (M23). SonarQube also has an underlying mapping, which maps different issues to an expected time to fix it. This cumulated time is used as an additional metric (M24).

$$\text{M23} = \frac{\text{SonarQube issues} * 1000}{\text{Logical lines of code (M15)}}$$

Metric 23: SonarQube Issues per 1000 Logical Lines of Code

$$\text{M24} = \text{Estimated time to fix all issues detected by SonarQube}$$

Metric 24: SonarQube Fix Time

3.1.5 Quantify Observations of Project A

The observations of project A made in section 3.1.2 are quantified in the following. The metrics are based on the GQM-model described in section 3.1.4. The concrete measurement results for the first goal (prevent scope creep) can be found in table 3.2. The results of the second goal can be found in table 3.3.

Goal 1: Interpretation of results

The questions of the first goal are ordered by the level of detail. The first question gives a rough overview if the goal was reached. The subsequent questions can be used to get information about the root causes. In other words, the first question indicates if the goal was reached and the subsequent questions indicate why the goal was or was not reached.

Table 3.2: G1 prevent scope creep: Measurement results project A

ID	Description	A
Q1.1	Was the project delivered in-time?	
M1	Estimated Project Duration [days]	64
M2	Schedule Estimation Accuracy	2.06
Q1.2	Was the project built within budget?	
M3	Estimated Project Effort [hours]	243
M4	Effort Estimation Accuracy	2.79
Q1.3	How were resources used?	
M5	Actual Project Effort [hours]	679
M6	Feature Effort Proportion	55 %
M7	Change Request Effort Proportion	37 %
M8	Failure Effort Proportion	8 %
Q1.4	How were resources used?	
M3	Estimated Project Effort [hours]	243
M9	Feature and Failure Effort to Estimation Ratio	1.55
M10	Feature Effort to Estimation Ratio	1.77
Q1.5	Were the change requests paid by the customer?	
M11	Billed Change Request Effort Proportion	3.5 %
Q1.6	Was the team successful in building trust by providing a reliable software in-time to put the project manager in a good change requests negotiating position?	
M12	Period of Observation [days]	173
M13	Total Failures Found	112
M14	Customer Found Failure Rate	19.4
M2	Schedule Estimation Accuracy	2.06

As metric 2 shows, the project took more than twice as long as arranged with the customer. Also the internal budget was highly exceeded (see figure 3.2). The effort of the team was nearly three times higher than expected. These metrics show that time and budget constraints were clearly missed.

Now the question is if the reason for that is scope creep as observed. At first, the work distribution on different task types is taken into account (see figure 3.2). It shows that more than a third of the whole resources were spent on change requests, whereas only eight percent were spent on failures, which seems to be ok. This would indicate, that the high number of change requests caused the project to fail. However, metrics M9 and M10 show, that the project exceeded the budget nearly

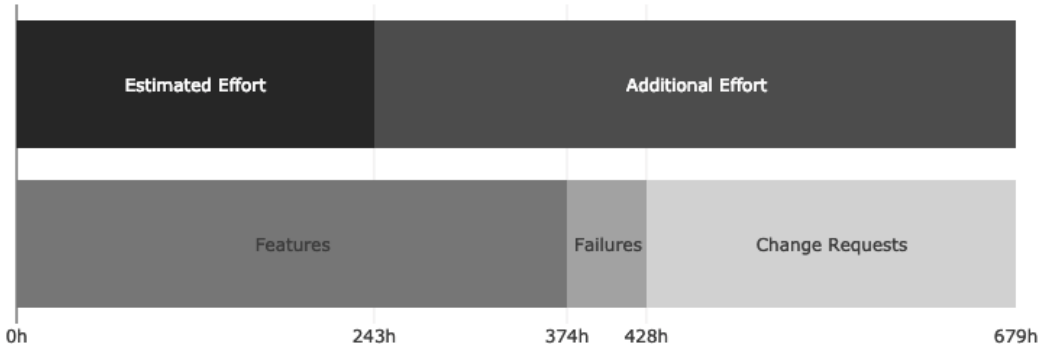

Figure 3.2: Project A: Effort distribution

twice even if there would have been no change requests at all. Even subtracting the fixing time of defects would not change the result. Therefore the first thing to mention is, that the initial estimation of the predetermined features was wrong. The second thing to mention, the high amount of change request indicates that the features were not chosen properly. This would not cause problems if the customer paid the change requests. However, metric M11 shows, that only three and a half percent of the effort regarding change requests could be billed. The suspicion that a poor specification caused the project to fail appears to have been confirmed.

However, why was the project manager not able to bill the additional effort caused by change requests? Question 1.6 and the corresponding metrics give a clear answer to that. Beside the fact, that the project was heavily exceeding the deadline, the customer's trust could not have been high, because the number of failures

found by the customer was high. One hundred twelve failures in total or rather 19.4 failures per month found by the customer is too much.

To sum up, the goal of preventing scope creep was clearly missed. A dynamic of exceeding a deadline, therefore working under pressure, which leads to bad code quality and therefore a bad position for change request negotiations was not prevented. The root cause was bad requirements engineering. Concrete actions to prevent scope creep are discussed in section 3.1.7.

Goal 2: Interpretation of results

Question 1.6 discussed in section 3.2.4 shows, that too many failures occurred. The presumption is that the high amount of change caused a high amount of technical debt. This could be a reason for the high amount of defects. To verify that assumption, the questions Q2.1 to Q2.4 target clean code.

The problem of the system is that no automated test suite is provided. That

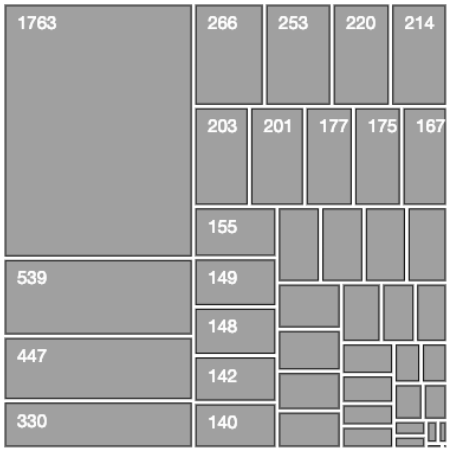

Figure 3.3: Project A: Distribution of lines of code on files (7185 total lines of code distributed on 43 files)

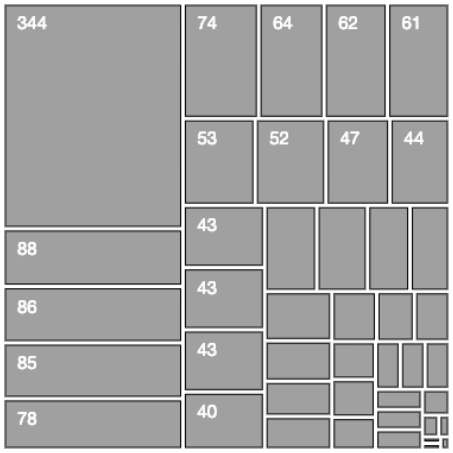

Figure 3.4: Project A: Distribution of cyclomatic complexity on files (Total complexity of 1690 distributed on 43 files)

makes understanding and changing code more difficult. The complexity of the

code is measured with lines of code and cyclomatic complexity. In figure 3.3 the distribution of the code lines on the different files is presented as a treemap. As one can see, there exists one file with 1800 lines of code. This file represents a monolith, which is hard to maintain. In addition to that, it handles much important functionality.

In figure 3.4 the treemap of the cyclomatic complexity is shown. The monolith also occurs in this figure with a cyclomatic complexity of 344. Measuring cyclomatic complexity in addition to the lines of code does not give deeper insights in this case. The cyclomatic complexity will be more expressive if it is compared to the other projects later on. At this point the recommendations for different cyclomatic complexity values by Steve McConnel can help to get a feeling for those values:

- 0-5: The routine is probably fine.
- 6-10: Start to think about ways to simplify the routine.
- 10+: Break part of the routine into a second routine and call it from the first routine.

It is important to say, that the metrics M19 and M20 correspond to the complexity per file and not to the complexity per routine. The complexity per file is, of course, greater or equal to the complexity per function, because it is the sum of the complexity of all functions in that file.

As mentioned above, the values will be more expressive in comparison to the other projects. What can be stated at this point is, that the maximum complexity of 344 seems to be very high. The percentage of duplicated lines is also high.

The metrics M23 and M24 are already explained in more detail in section 3.1.4. The values measured are without any doubt not good. For every 1000 lines of code, 140 issues are introduced. It is estimated that it would take 164 hours to remove those issues.

To conclude, there are some serious anomalies in the metrics describing the maintainability of the code. As observed and suspected, corrective actions are necessary for the next project. Before elaborating these actions, project B is described in detail.

Table 3.3: G2 deliver maintainable code: Measurement results project A

ID	Description	A
Q2.1	How complex is the code?	
M15	Lines of Code	7185
M16	Maximum Logical Lines of Code	1800
M17	Average Logical Lines of Code	167.1
M18	Cyclomatic Complexity	1690
M19	Maximum Cyclomatic Complexity	344
M20	Average Cyclomatic Complexity	39.3
Q2.2	Is the code well documented by a meaningful test suite?	
M21	Appropriate automated test suite applied	No
Q2.3	How hard is it to change the code?	
M22	Percentage of Duplicated Lines	6.8 %
M21	Appropriate automated test suite applied	No
Q2.4	Is the amount of technical debt acceptable?	
M23	SonarQube Issues per 1000 Lines of Code	140
M24	SonarQube Fix Time [hours]	164
M22	Percentage of Duplicated Lines	6.8 %

3.1.6 Description of Project B

Project B extends the product of project A. Therefore the technologies used are PHP in combination with Concrete 5 as the content management system for the backend and JavaScript for the client-side application. The product helps to schedule and manage courses, instructors and participants. The team setting was the same as at the end of project A. The project manager, which is the author of this work, was also contributing to the client-side application and his colleague was working on the backend. Of course, the fact that this project is not written from scratch, that the team did not change from project A to project B, that the team is now more experienced with the used technologies and that the team was already familiar with the code base, could have a significant impact on the results. This issue is therefore discussed at the end of this iteration (see section 3.1.8) and in the final discussion (see section 5).

3.1.7 Corrective Actions

The goals of this iteration are preventing scope creep and delivering maintainable code. Which actions are chosen, why they are chosen and in which way they should help, is discussed in the following.

Preventing scope creep

In section 3.1.2 it is already mentioned, that better requirements development and management practices in addition to a good change request process would most likely lower the risk of scope creep. This assumption correlates with literature. Many publications blame poor requirements engineering practices to jeopardize project success [37][88][71][50]. Additionally Alcides Quispe et al. show that poor requirements engineering practices are common and have disastrous consequences on project success, especially in very small companies [68]. Their findings indicate, that

1. project specifications are usually met, but the client often finds the solution unsatisfactory.
2. communication issues with clients cause incomplete specifications.

3. the project's scope expands as clients require additional changes
4. requirements specification in VSSEs is mostly an ad-hoc process.
5. this ad-hoc process leads to requirement management issues such as loss of requirements.
6. when uncertainty arises, developers tend to resolve the issue without contacting the clients.
7. very small enterprises are aware of the benefits of requirements engineering practices but are not sure they apply in their context.

This enumeration represents the problems of project A. Requirements engineering is also mentioned to be a key factor to increase the success rate of software projects [26][51][63].

In terms of project risk, Mark Keil et al. introduced a framework [45], which consists of four quadrants shown in figure 3.5. The four areas result from two dimensions:

- Seriousness of the risk
- Level of control of the project manager on the risk itself

Scope and requirements are located in the top right quadrant. They have high importance and are well controllable. This assumption was verified by a survey with 507 participating project managers [90]. The outcome is that the scope and the requirements are of high importance. In addition to that, they found that the combination of high execution risk, e.g., low experience of the project team, with high requirements and scope risk has a nine times higher influence on the outcome of the project as requirements and scope risks alone. They also give some guidance:

> Practically speaking, this means that project managers who know execution risk is high and are unable to lower it must develop a risk-mitigation strategy focusing on minimizing the risks associated with scope/requirements and customer mandate.

The CMMI also indicates that requirements management is of high priority. The CMMI model consists of twenty-two process areas and five maturity levels. Seven of those areas have the lowest maturity level. Requirements management is one of them whereas requirements development is a process area of the next maturity level [21].

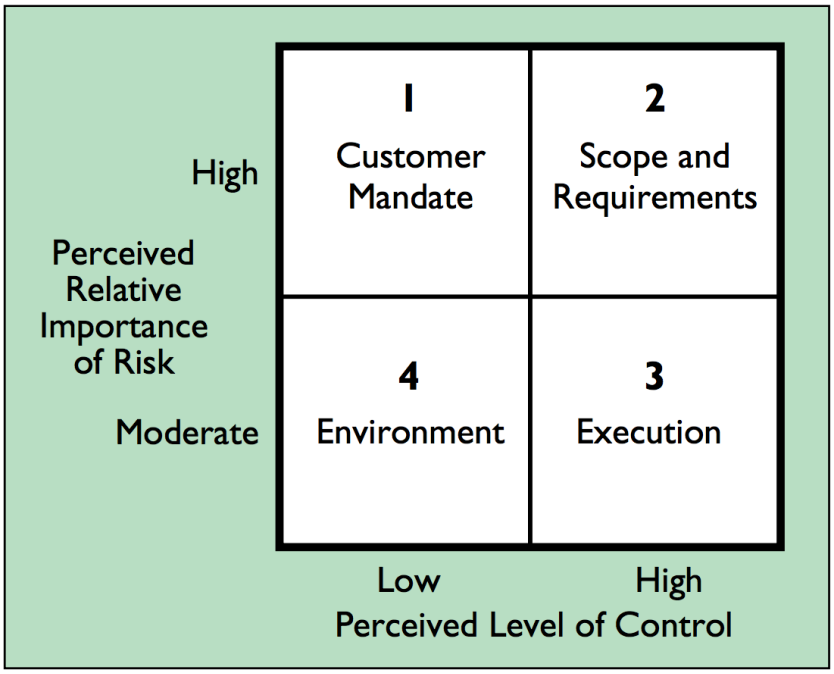

Figure 3.5: Project risk framework [45]

Therefore a requirements development and management process will be introduced. The actual process is described at the end of this section.

Delivering maintainable code

The practice of code reviews is described for the first time by Michael Fagan in 1976 [28]. He described a formal process including a design- and a code review. Nowadays, code reviews are used differently. Therefore the term ”Modern Code Reviews” was introduced, which describes code reviews as informal and tool-based [6].

Code reviews are primarily introduced to find defects in code. Therefore, code reviews would not target the goal of delivering maintainable code. However, firstly research showed, that the expectations of doing code reviews do not fully correlate with the actual outcomes and secondly code improvements are the second most desired effects of code reviews [6]. Additionally, knowledge transfer and distribution is desired.

Bacchelli and Bird showed that most of the comments provided by code reviews target code improvements. They think that this is caused by the fact, that these improvements are easy to submit, whereas finding defects is much more challenging because a deep understanding of the code by the reviewer has to be attained [6]. Summarizing, modern code reviews primary outputs are code improvements regarding

- better code practices,
- removing not necessary or unused code and
- improving code readability.

Therefore introducing modern code reviews is a good choice towards the goal of increasing maintainability. The concrete implementation of the practice is described in the following.

Introducing the process

The actions chosen will be combined and integrated into a defined process. The new process is shown in figure 3.6 and described in the following.

Requirements Development The process includes a part for requirements development. The detailed specification of a feature is done by a product owner (S1). The product owner passes the specification to the technical lead, who estimates the effort (S2). The elaborated and estimated feature is reviewed and declared as ready for development by the customer itself (S3). If the technical lead or the customer has problems in understanding the feature described, the feature is passed back to the product owner (S1). The customer also has the possibility to deny a feature, which leads to an end state (S9).

Implementation If a feature is ready for development (S4), it is the responsibility of the project manager to assign developers and keep track of the progress. As code reviews are part of the new process, the development team uses another branching strategy. In project A, everyone was working directly on the mainline. In project B the branching model ”Branch-by-feature” is used. Therefore the developer creates a branch from the mainline to work isolated on the feature. After implementing the feature, the developer creates a pull request and assigns the technical lead to the feature (S6). If there are no problems detected, the changes are merged into mainline and the feature is passed to the internal quality assurance (S7). If problems were detected, they are documented, and the ticket is passed back to ”Work in Progress”. It is the responsibility of the developer to adjust the changes.

Acceptance The mainline is then deployed to a testing environment and manually tested by the internal quality assurance. In case of problems, the feature can be rejected. Otherwise the mainline is deployed to the staging environment and reviewed by the customer. The customer is then able to accept the feature or to reject it. In the case of rejection, the project manager is responsible for distinguishing, if it is a failure or a change request. A failure would cause the feature to get back work in progress (S5). If the claim represents a divergence

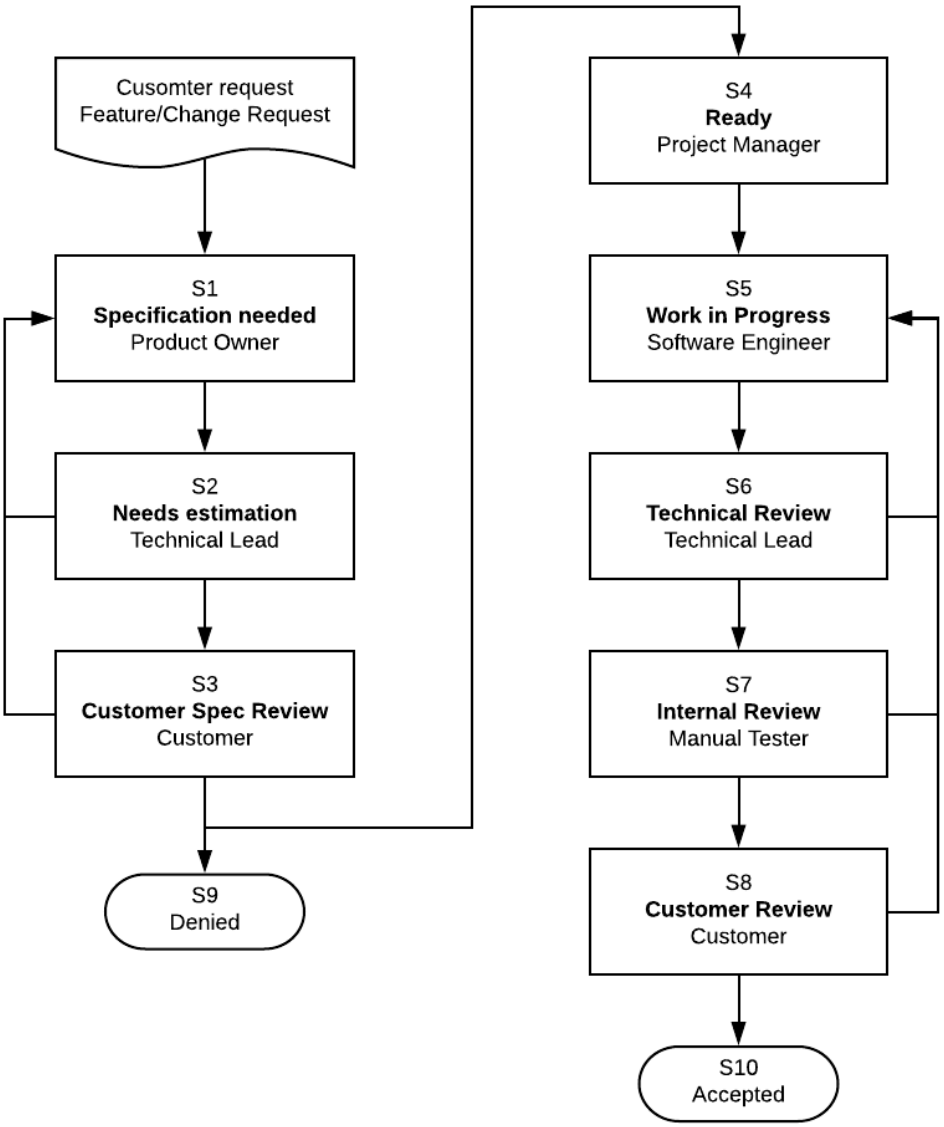

Figure 3.6: Elaborated process

to the accepted specification, the new desired functionality is treated as a change request. In this case, the change request is treated like every customer request and has to go through the whole process as every feature does. The original feature is then treated as accepted (S10).

3.1.8 Analyze Impact on Project B

As already mentioned, project B was an extension of project A. It added new functionality with a nearly equally high estimated effort as the base project itself. This makes requirements engineering easier because all parties already know the project itself.

Tables 3.4 and 3.5 show the measurable impact of the corrective actions described in section 3.1.7. The tables show the concrete results of project A and B side by side. Goals in the following provide a detailed analysis of those values.

Goal 1: Preventing scope creep

The analysis starts with question 1.1, if the project was delivered in-time. Project B also exceeds the deadline as shown in figure 3.7. But project B exceeded the deadline only by 59 percent (M2). This is a significant improvement to 106 percent in project A.

The effort estimation is topic to question 1.2. Figure 3.9 shows, that project B was still more effort than estimated, but 15 percent (M4) is an acceptable value, especially compared to 179 percent in project A. This comparison is also shown in figure 3.8.

By looking at the effort and the duration of the project one sees significant improvements to project A. Therefore the goal of preventing scope creep can already be considered as reached. The reason is caused by less change requests. The effort distribution of project B is shown in figure 3.9. Figures 3.10 and 3.11 show the change of the change request's proportion from project A to project B. These figures show, how the resources were used (question 1.3, M6, M7, M8).

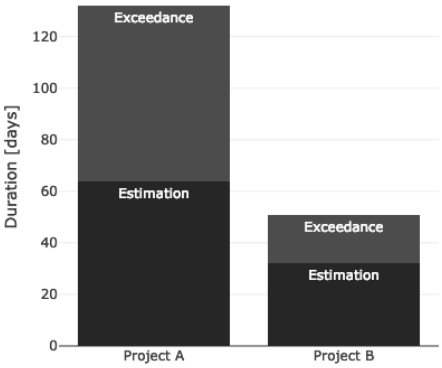

Figure 3.7: Project duration

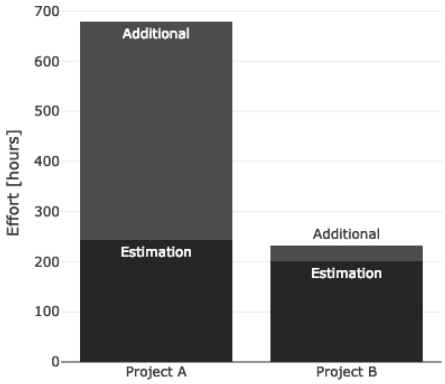

Figure 3.8: Project effort

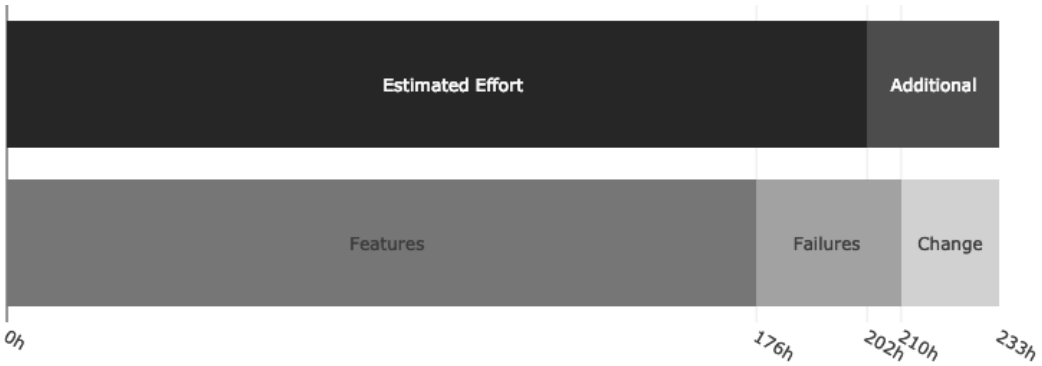

Figure 3.9: Project B: Additional effort and effort distribution

Question 1.4 targets feature estimation. Metrics M9 and M10 show, if the project is built within estimated effort if the time spent on failures respectively the time spent on failures and change requests is neglected. The metrics show, that the estimation was already nearly correct by only four percent additional effort when change requests are neglected. Effectively this can be done because 86 percent of the change requests were paid by the customer (M11). In project A only three and a half percent of the change requests were paid. By also neglecting the time spent on failures or respectively only looking at the time spent on real features, the

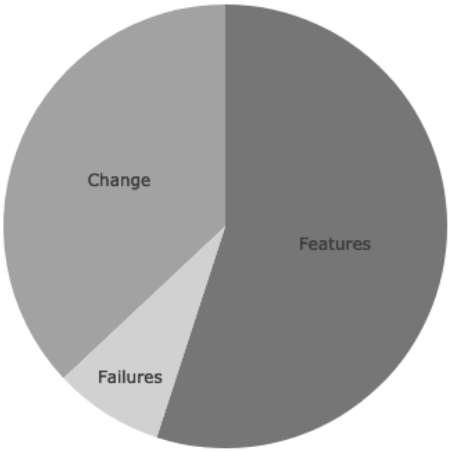

Figure 3.10: Project A: Effort distribution

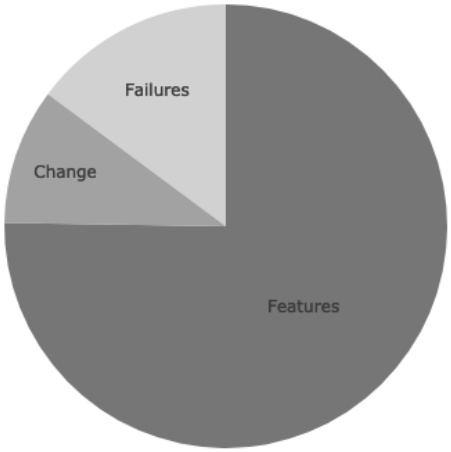

Figure 3.11: Project B: Effort distribution

effort met the estimation. This set of metrics is also visualized in figure 3.9. The lower bar shows the effort of the different types of issues. By comparing those to the upper bar, one can see, that the time spent on features is lower than the effort estimation and also the time spent on features and failures exceeds the estimation just a little bit.

As already mentioned, nearly all change requests were paid by the customer. This is most likely due to a decreased customer found failure rate (M14). There were only six and two tenth failures found per month compared to nineteen and four tenths in project A. In addition to that, and as already mentioned, project B was delivered earlier as its predecessor (M2).

Goal 2: Delivering maintainable code

As code reviews were used to keep the code maintainable, it is important to state, that only new checked in code was reviewed. As already mentioned, this project is an extension. There was still no test suite, and the monolith, which caused the maximum lines of code per file and the maximum cyclomatic complexity per file is still in the code, and the metrics even increased (see figure 3.12 and 3.13). The results are therefore not as meaningful as desired.

Table 3.4: G1 prevent scope creep: Measurement results project A vs. project B

ID	Description	A	B
Q1.1	Was the project delivered in-time?		
M1	Estimated Project Duration [days]	64	32
M2	Schedule Estimation Accuracy	2.06	1.59
Q1.2	Was the project built within budget?		
M3	Estimated Project Effort [hours]	243	202
M4	Effort Estimation Accuracy	2.79	1.15
Q1.3	How were resources used?		
M5	Actual Project Effort [hours]	679	233
M6	Feature Effort Proportion	55 %	76 %
M7	Change Request Effort Proportion	37 %	10 %
M8	Failure Effort Proportion	8 %	15 %
Q1.4	How were resources used?		
M3	Estimated Project Effort [hours]	243	202
M9	Feature and Failure Effort to Estimation Ratio	1.55	0.87
M10	Feature Effort to Estimation Ratio	1.77	1.04
Q1.5	Were the change requests paid by the customer?		
M11	Billed Change Request Effort Proportion	3.5 %	86 %
Q1.6	Was the team successful in building trust by providing a reliable software in time to put the project manager in a good change requests negotiating position?		
M12	Period of Observation [days]	173	213
M13	Total Failures Found	112	44
M14	Customer Found Failure Rate	19.4	6.2
M2	Schedule Estimation Accuracy	2.06	1.59

However, as the proportion of duplicated lines (M22) and the issues per 1000 lines of code (M23) decreased, it seems, that the code changes improve the cleanness of the code. This improved cleanness is an indicator that the code reviews had a positive impact. Metrics M16, M17, M19 and M20 can therefore nevertheless be worse due to the fact, that the project was an extension and an increasing complexity was not preventable but at least kept low. Figures 3.12 and 3.13 represent a bar chart, whereas the individual bars are presented as treemaps. As the width of each bar is the same, the height is directly proportional to the total sum of code lines respectively the total cyclomatic complexity. One can still see the monoliths. However, by comparing the proportions between project A and B regarding cyclomatic complexity and lines of code, an interesting observation can be made. The lines of code increased more than the complexity did. The total lines of code increased by 24.8 percent, whereas the total complexity only increased by 17.5 percent. This indicates that the code base got less complex compared to its size.

To sum up, question Q2.1 shows that the complexity of the code increased, possibly because the project was an extension. Project C will give more meaningful insights. Questions Q2.2 and Q2.3 show that there is no significant improvement in the understandability and changeability of the code. Maybe also caused by the original code base. At last question Q2.4 shows, that the technical debt was decreased. The results, therefore, are not conclusive. If the code reviews really succeeded and therefore delivered the desired outcomes will be clarified by project C, which will cover a new product, which is built from scratch.

Threats to validity

As already mentioned in the description of project B (see section 3.1.6), the team that finished project A, was also the team working on project B. In addition to that, the code base was the same, as the created product in project A was extended in project B. In the following, it is presented in which way these facts and others could have had an impact on the gathered results. In this case, it is taken advantage of the fact, that the author of this work was operatively involved in both projects. His qualitative perception of the situation can help to determine, which results are really caused by the applied actions.

Table 3.5: G2 deliver maintainable code: Measurement results project A vs. project B

ID	Description	A	B
Q2.1	How complex is the code?		
M15	Lines of Code	7185	8964
M16	Maximum Logical Lines of Code	1800	1990
M17	Average Logical Lines of Code	167.1	190.7
M18	Cyclomatic Complexity	1690	1987
M19	Maximum Cyclomatic Complexity	344	424
M20	Average Cyclomatic Complexity	39.3	42.3
Q2.2	Is the code well documented by a meaningful test suite?		
M21	Appropriate automated test suite applied	No	No
Q2.3	How hard is it to change the code?		
M22	Percentage of Duplicated Lines	6.8 %	2.6 %
M21	Appropriate automated test suite applied	No	No
Q2.4	Is the amount of technical debt acceptable?		
M23	SonarQube Issues per 1000 Lines of Code	140	94
M24	SonarQube Fix Time [hours]	164	141
M22	Percentage of Duplicated Lines	6,8 %	2.6 %

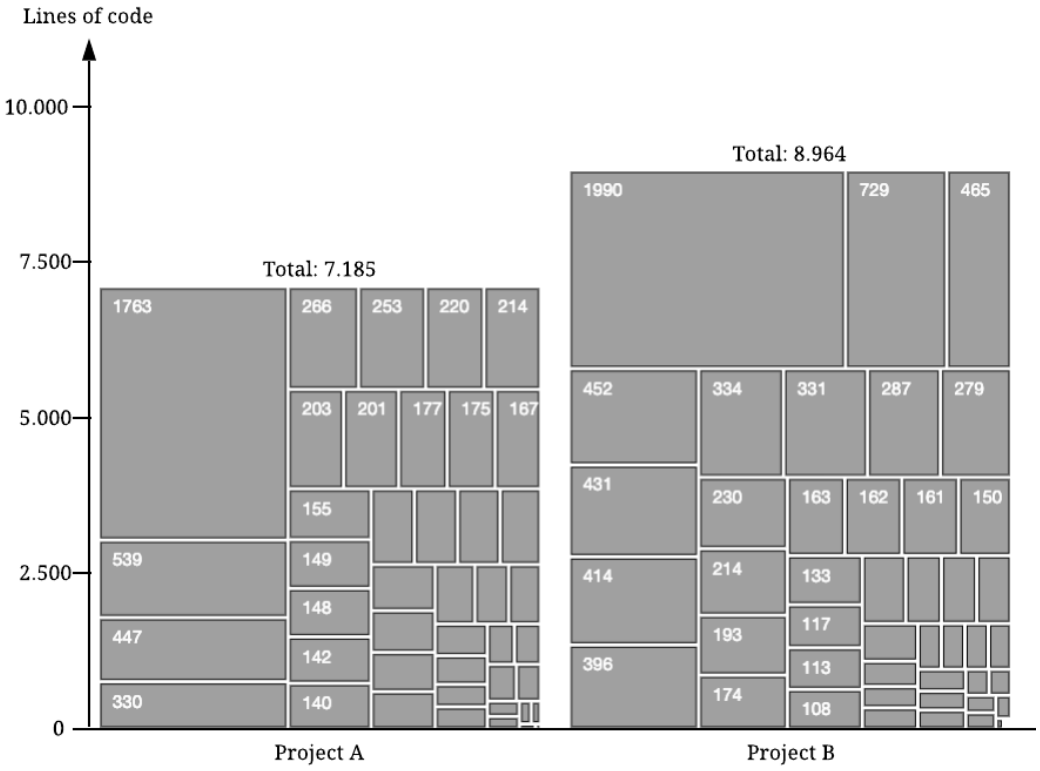

Figure 3.12: Tree map of lines of code of projects A and B

The prevention of scope creep was the first goal of this iteration. The main result was, that project A exceeded the estimated effort by 179 percent and project B by only 15 percent. By reflecting both projects, this tremendous difference was mainly obtained by better requirements engineering. However, it was also supported by the constellation of this iteration, because the estimation and the requirements engineering were easier for project B in comparison to a project, which creates a product from scratch. This important parameter is not present in project C, as the domain, the customer, and the technologies used are different to project A and B. The second iteration will give deeper insights about the impact of knowing the product and the domain already. The results show with no doubt that well-defined requirements prevent scope creep. The remaining question is if the requirements engineering process can also produce such well-defined requirements in an un-

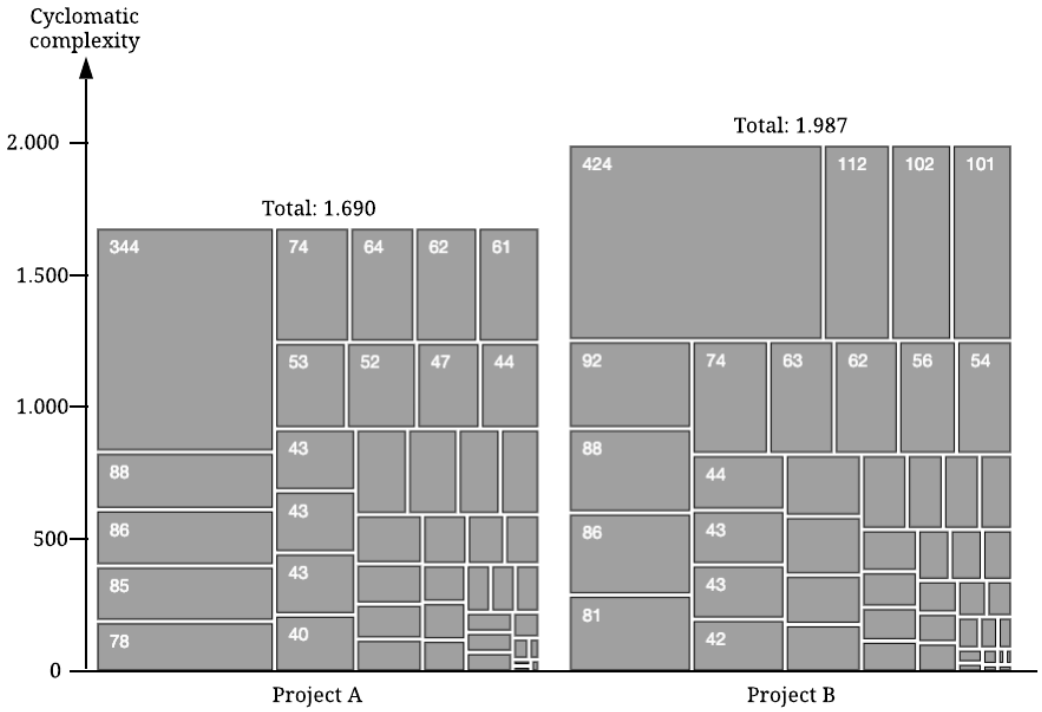

Figure 3.13: Tree map of cyclomatic complexity of projects A and B

known domain.

Regarding the modern code reviews, it is already mentioned, that a conclusive validation cannot be obtained by analyzing the extension of a product. A slight decline of the overall technical debt indicates success, but project C will give more appropriate evidence.

3.2 Iteration 2

As in the first iteration, each of the used methodology's six steps has its subsection in the following. In addition to that, a description of project C is provided in its own subsection.

3.2.1 Observations of Project B

The goal of project B was extending the web application, which was the output of project A. As described, the product serves as a tool for scheduling courses automatically and for supporting the management of instructors and participants. The details of project B are already described (see section 3.1.6). As this is not the first iteration, the analysis of the first iteration can be used in addition to subjective observations.

By having a look on tables 3.4 and 3.5 some observations can already be derived. Metric M4 shows, that project B still overran the proposed budget slightly. Also, the deadline was missed slightly (M2).

> Observation B-1: Still little problems with delivering in-time and within a predetermined budget

It was observable that the development of features worked well. It seems that the still late acceptance is caused by the work on change requests and failures. Changing the system is a high effort especially for the manual quality assurance. Of course, this effort is exponentially higher with no test suite applied. There are two choices in developing software:

- No test suite applied and therefore a low delivery frequency in respect to a high manual quality assurance effort or
- a high delivery frequency, only possible with an appropriate automated test suite.

Problems occur when the delivery frequency is high, and no automated test suite is applied. Problems of this kind could be observed in project B. As metric M21 already shows, there is still no test suite applied. The team was developing with a good pace and finished features on time. The manual quality assurance was looking for issues regarding functionality and performance. Everything was fine up to that point in time. The delivery frequency was low and therefore having no test suite did not cause problems. The situation changed when change requests and failures found by the customer occurred. In this situation, the delivery frequency increased, as every change request and especially every fix should have been delivered as soon as possible. Theoretically, the manual quality assurance would have had to check the whole system each time before a new version was delivered to the

customer. On the one hand cycle and lead times of change requests and fixes were long. On the other hand, the increasing pressure on manual quality assurance led to faster checks and therefore to a lower delivered quality. As a result, more failures found by the customer occurred, and therefore the pressure caused by a dissatisfied customer rose. The following observations can be derived:

> Observation B-2: Still no test suite

> Observation B-3: Everything works fine until the moment of an increasing delivery frequency

> Observation B-4: Unmanageable effort for manual quality assurance

> Observation B-5: High lead and cycle times of changes and fixes

Of course, this scenario leads to great discomfort in releasing new versions. Developers are afraid of breaking the system, the quality assurance has an enormous effort, and the project manager has problems with the customer due to quality issues.

> Observation B-6: High discomfort in releasing new versions

The last noticeable thing is that it is yet not clear if the code reviews had a positive impact on code quality. The percentage of duplicated lines (M22) decreased and the estimated maintenance effort (M23, M24) improved, whereas all other code quality metrics slightly worsened (M16, M17, M19, M20). Interestingly, the complexity did not grow as much as the lines of code did. This is already a good sign. As project B was an extension of project A and therefore not written from scratch, the question on the impact of code reviews cannot be answered definitely.

> Observation B-7: Impact of code reviews yet not clarified - ambiguous results

In table 3.2.1 the observations are collected.

Table 3.6: Observations of project B

ID	Description
O-B-1	Still little problems with delivering in-time and within predetermined budget
O-B-2	Still no test suite
O-B-3	Everything works fine until the moment of an increasing delivery frequency
O-B-4	Unmanageable effort for manual quality assurance
O-B-5	High lead and cycle times of changes and fixes
O-B-6	High discomfort in releasing new versions
O-B-7	Impact of code reviews yet not definitely clarified - ambiguous results

3.2.2 Define Goals

The observations ultimately describe one goal: A schedule estimation accuracy of 1 (M2). Observations O-B-1 to O-B-6 are already reasons, why the project is not accepted by the customer earlier. O-B-7 is already targeted by goal 2 and needs to be validated in a new project that is written from scratch.

3.2.3 Extend GQM-Model

The goal of this section is to extend the GQM-Model by a third goal, namely: ”Achieve a schedule estimation accuracy of 1”. In the following, the related questions and metrics are elaborated to extend the GQM-Model. This will be used to determine, how the concrete actions affected the project outcomes objectively.

Goal 3: Achieve a schedule estimation accuracy of 1

The first question to be asked is if the project was delivered in-time. The following questions help to identify the reasons for a late acceptance and will be part of the GQM-model for goal 3:

- Q3.1: Was the project delivered in-time?

- Q3.2: Did a wrong effort estimation cause the delay of acceptance?
- Q3.3: Was the planning of resources appropriate?
- Q3.4: How much of the work was done from due day to day of acceptance?
- Q3.5: How was the time used from due day to day of acceptance?
- Q3.6: How was the time used until the due day?
- Q3.7: How easy was it to deliver change requests and fixes?

First of all, it needs to be verified, if the observations are the actual root causes. The assumption is that the development of features was finished before the due day and most of the work done after due day was caused by change requests and fixes. This would indicate, that no estimation or resource planning mistakes occurred. This is verified by the questions Q3.2 to Q3.6.

If effort estimation and resource planning were done right, the suspicion is, that it was hard to deliver changes and fixes after the development of features was complete. This suspicion is verified by question 3.7.

Metrics for Q3.1 - Was the project delivered in-time? This question is the same as Q1.1. For a detailed description see section 3.1.4 and metrics M1 and M2.

Metrics for Q3.2 - Did a wrong effort estimation cause the delay of acceptance? To answer this question, effort estimation accuracy (M4) is taken into account. It shows, if and how much the invested effort exceeded the estimated effort. For better comparability, the total estimated effort is also given in hours (M3).

Metrics for Q3.3 - Was the planning of resources appropriate? The human resources are allocated in the planning phase of a project. If this is done right, the invested hours until the due day should not be lower than the estimated effort, expect the total invested effort was lower than the estimated effort. Therefore the invested hours until due day divided by the estimated effort is used to answer this question (M25). In addition to that, the total estimated effort (M3) is provided for better comparability.

$$\text{M25} = \frac{\text{Invested hours until due day}}{\text{Estimated project effort in hours (M3)}}$$

Metric 25: Invested hours by estimated hours until due day

A value of one means that the planning was proper. However, if the project exceeded the deadline, the estimation caused the problems. A value higher than one means that the project manager already reacted on the bad effort estimation. A value lower than one is ok if the project was delivered in-time. Otherwise, it means, that the resource planning failed. Of course, it is also possible that the estimation and resource planning was bad.

Metrics for Q3.4 - How much of the work was done from due day to day of acceptance? The time until the due day has a low delivery frequency. The reason for asking this question is to find out how much effort of the project was made under pressure when the project was already delayed. The time after the due day has a higher delivery frequency and is predestinated for introducing defects. This question shows how significant that proportion is. Therefore three metrics are defined. M26 and M27 represent the absolute values, whereas M28 shows the percentage of time spent from due day to day of acceptance.

$$\text{M26} = \text{Total hours spent until due day}$$

Metric 26: Total hours spent until due day

$$\text{M27} = \text{Total hours spent from due day to day of acceptance}$$

Metric 27: Total hours spent from due day to day of acceptance

$$\text{M28} = \frac{\text{Total hours spent from due day to day of acceptance (M27)}}{\text{Time spent since due day (M27)} + \text{Time spent until due day (M26)}}$$

Metric 28: Invested hours by estimated hours until due day

Metrics for Q3.5 - How was the time used from due day to day of acceptance? This question is very insightful. If the team worked solely on fixes and change requests, at least the resource planning was good. If the team was still working on features after due day, the project was not planned well. Therefore the proportions of hours worked on features (M29), hours worked on failures found by customer (M30) and hours worked on change requests (M32) from due day to day of acceptance are given. In addition to that, the failures found by internal quality assurance will get tracked in project C. Also this proportion is taken into account (M31). The denominator is the total time spent from due day to day of acceptance (M27).

$$\text{M29} = \frac{\text{Total hours spent on features since due day}}{\text{Total hours spent from due day to day of acceptance (M27)}}$$

Metric 29: Prop. of hours worked on features since due day

$$\text{M30} = \frac{\text{Total hours spent on failures found by customer since due day}}{\text{Total hours spent from due day to day of acceptance (M27)}}$$

Metric 30: Prop. of hours worked on failures found by customer since due day

$$\text{M31} = \frac{\text{Total hours spent on failures found by QA since due day}}{\text{Total hours spent from due day to day of acceptance (M27)}}$$

Metric 31: Prop. of hours worked on failures found by QA since due day

$$\text{M32} = \frac{\text{Total hours spent on change requests since due day}}{\text{Total hours spent from due day to day of acceptance (M27)}}$$

Metric 32: Prop. of hours worked on change requests since due day

Metrics for Q3.6 - How was the time used until the due day? If the time after due day was spent solely on change requests and fixes, it is interesting, if the team was already working on change requests and fixes before the due day. This would indicate, that the estimation was good, but maybe a bad quality led to a high effort of fixing defects or maybe poorly defined requirements caused a high amount of change requests. Therefore, as in question 3.5, the proportions of hours worked on features (M33), hours worked on failures found by customer (M34), hours worked on failures found by internal quality assurance (M35) and hours worked on change requests (M36) until due day are investigated. In this case, the total hours spent until the due day (M26) are used as the denominator.

$$\text{M33} = \frac{\text{Total hours spent on features until due day}}{\text{Total hours spent until due day (M26)}}$$

Metric 33: Prop. of hours worked on features until due day

$$\text{M34} = \frac{\text{Total hours spent on failures found by customer until due day}}{\text{Total hours spent until due day (M26)}}$$

Metric 34: Prop. of hours worked on failures found by customer until due day

$$\text{M35} = \frac{\text{Total hours spent on failures found by QA until due day}}{\text{Total hours spent until due day (M26)}}$$

Metric 35: Prop. of hours worked on failures found by QA until due day

$$\text{M36} = \frac{\text{Total hours spent on change requests until due day}}{\text{Total hours spent until due day (M26)}}$$

Metric 36: Prop. of hours worked on change requests until due day

Metrics for Q3.7 - How easy was it to deliver change requests and fixes?
If the suspicion that the team was struggling with the high delivery frequency combined with the non-existence of a test suite is confirmed, the lead and cycle times of fixes and change requests are important to be examined. The cycle time begins with the start of the work and ends with the delivery to the customer. The lead time consists of the wait time from the customer's request to the start of the work and the cycle time. As the data set will most likely contain outliers, the median values are used (M37 and M39). In addition to that, box plots will be used to visualize the cycle and lead times. Therefore the interquartile ranges (IQR) are given (M38 and M40). Fifty percent of the samples are within the interquartile range per definition.

M37 = Median lead time for fixes and change requests in calendar days

Metric 37: Median lead time for fixes and change requests

M38 = IQR of lead times for fixes and change requests in calendar days

Metric 38: IQR of lead times for fixes and change requests

M39 = Median cycle time for fixes and change requests in calendar days

Metric 39: Median cycle time for fixes and change requests

M40 = IQR of cycle times for fixes and change requests in calendar days

Metric 40: IQR of cycle times for fixes and change requests

3.2.4 Quantify Observations of Project B

The observations of project B made in section 3.2.1 are quantified in the following. The metrics are based on the GQM-model described in section 3.2.3. The concrete measurement results for the third goal, namely achieving a schedule estimation accuracy of 1, can be found in table 3.7.

Goal 3: Interpretation of results

In this section, the observations of project B are quantified. For that purpose, it would suffice to look only on the values of project B. As the metrics also changed a lot from project A to project B, the values of project A are also provided. Those values give additional insights about the effects of the corrective actions of iteration 1. These effects are therefore also discussed subsequently.

Table 3.7: G3 achieve a schedule estimation accuracy of 1: Results of projects A and B

ID	Description	A	B
Q3.1	Was the project delivered in-time?		
M1	Estimated Project Duration [days]	64	32
M2	Schedule Estimation Accuracy	2.06	1.59
Q3.2	Did a wrong effort estimation cause the delay of acceptance?		
M3	Estimated Project Effort [hours]	243	202
M4	Effort Estimation Accuracy	2.79	1.15
Q3.3	Was the planning of resources appropriate?		
M3	Estimated Project Effort [hours]	243	202
M25	Invested hours by estimated hours until due day	1.57	0.79
Q3.4	How much of the work was done from due day to day of acceptance?		
M26	Total hours spent until due day	381	160
M27	Total hours spent since due day to day of acceptance	274	33.5
M28	Invested hours by estimated hours until due day	42 %	17 %
Q3.5	How was the time used from due day to day of acceptance?		
M29	Prop. of hours worked on features since due day	43 %	40 %
M30	Prop. of hours worked on failures found by customer since due day	15 %	16 %

M31	Prop. of hours worked on failures found by QA since due day	-	-
M32	Prop. of hours worked on change requests since due day	43 %	44 %
Q3.6	How was the time used until due day?		
M33	Prop. of hours worked on features until due day	66 %	99 %
M34	Prop. of hours worked on failures found by customer until due day	1.7 %	0.2 %
M35	Prop. of hours worked on failures found by QA until due day	-	-
M36	Prop. of hours worked on change requests until due day	32 %	0.6 %
Q3.7	How easy was it to deliver change requests and fixes?		
M37	Median lead time for fixes and change requests	-	7.66
M38	IQR of lead times for fixes and change requests	-	8.38
M39	Median cycle time for fixes and change requests	-	5.87
M40	IQR of cycle times for fixes and change requests	-	6.91

As already known, the project exceeded the deadline. Project B took 59 percent longer than expected (M2), whereas 15 percent more effort was required than estimated (M4). Therefore one knows, that the effort estimation did not cause the project duration to exceed. One reason for that can be found in the resource planning, because only 160 hours were worked until due day. That are only 79 percent (M25) of the estimated project effort. The combination of the metrics M28, M29 and M33 shows, that the deadline was already missed caused by poor resource planing. The argumentation works in the following way: 17 percent of the overall effort was done after due day and before acceptance (M28). This effort was not only used for change requests (44 percent, M32) and fixes (16 percent, M30), but also for working on features (40 percent, M29). And as metric M33 shows, in the time until due day the whole team was solely working on features (see figure 3.16). An appropriate resource planning could have led project B perform much better in terms of schedule estimation accuracy, because all the effort working on

features should have happened before due day and only 79 percent of the estimated effort was really performed before due day.

An additional criteria is the effort for working on fixes and change requests after due day. 60 percent of the time after due day was used to work on fixes and change requests (M30 and M32). The suspicion is, that those fixes and change requests have long cycle and lead times, caused by the non-existence of a test suite and therefore a high manual testing effort for the internal quality assurance for each delivery. The median cycle and lead times can be found in table 3.7. Beside that, the interquartile range is given. These values are illustrated in a box plot (see figure 3.14).

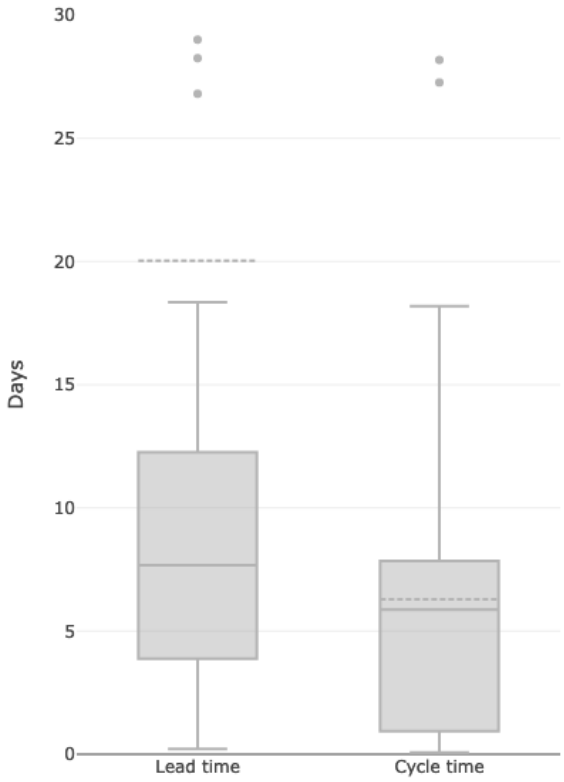

Figure 3.14: Lead and cycle times of project B

One can observe, that the median lead time is around seven and a half days (M37). Fifty percent of all values are between 3.87 and 12.26 days. The lead times are more dispersed than the cycle times. The dashed line shows the average and the solid line the median. As one can see, the median and average lead times are far away from each other in comparison to the corresponding cycle times. Two outliers of the lead time are not visible in the box plot, because they have a lead time of 162 respectively 267 days. To keep the box plot readable, the y-axis shows only 30 days. It seems that these outliers are caused by their wait times, because, as

already mentioned, the cycle times do not contain extreme outliers.

To interpret these values, they need to be examined concerning the agreed respectively the estimated project duration. The estimated project duration of project B was 32 days (M1). More concrete, if the customer finds a defect after due day, one out of two times, it will take about four to twelve days to fix that and to deliver it to the customer. As the acceptance of the project is tied to the removal of most of the defects found by the customer, these lead times seem to have a big impact on the project delay in addition to the problems in resource planning.

Additional insights in iteration 1

As already mentioned, some values for goal G3 are also measured for project A. This provides deeper insights about what happened in iteration 1.

In project A the company provided additional resources, because it was clear already early in the project, that the due day will be missed. More than 50 percent more resources than estimated were used until the due day (M25). The reason, why the due day was nevertheless clearly missed, was, that still 42 percent of the overall effort was done after due day and before acceptance. The usage of the time until the due day (question Q3.6) is another indicator, that scope creep occurred. In project A the team was already working on change requests before the due day. Only 66 percent of the time was used to work on features (M33). Of course, this would be desirable, if the team would have been finished with all features before. As this was not the case, figure 3.15 indicates, that the team was already faced with change requests early in the project. This shows that the specification was poorly written. In contrast to project A, the team was able to work 99 percent of the time on features until the due day in project B. The requirements engineering of project B was therefore successful. The difference is also presented in figures 3.15 and 3.16.

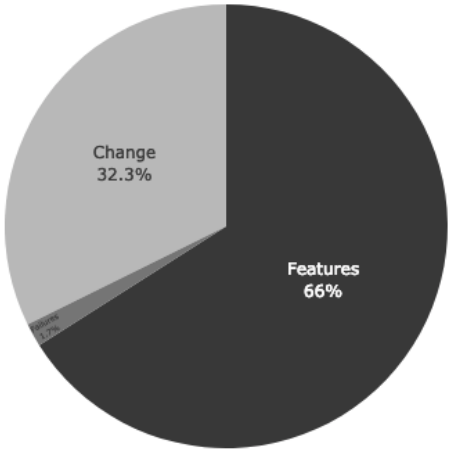

Figure 3.15: Project A: Effort distribution until due day

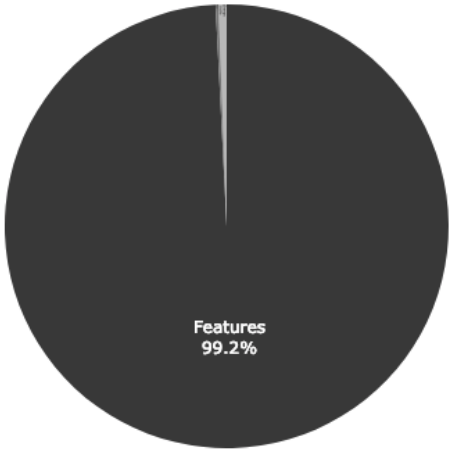

Figure 3.16: Project B: Effort distribution until due day

3.2.5 Description of Project C

Project C is more representative concerning impacts generated by the different corrective actions because it is a new application written from scratch ordered by another customer. In the first iteration, the customer and the application were the same on both reference projects. This issue is a threat to validity and is discussed in more detail in section 3.1.8. Project C provides a web service for a client-side application, which is developed by another company due to limited human resources. Node.js was used to implement the web service. In addition to that, a small configuration frontend was developed in-house by using Aurelia as frontend-framework. However, this configuration frontend was just a tenth of the effort of the web service itself. This part was developed by a full-time employee, which was already quite experienced with the use of Aurelia. Again the author of this work was part of the team in his role as project manager and backend developer, with a few months of experience with Node.js. He was supported by his colleague of project B, although the colleague had no experience with Node.js and backend JavaScript up to that point. These two were also the only persons, which were working full-time on project C. This core team was partially supported by the already mentioned frontend developer, by another developer, who set up the skeleton of the web service, what he already did a few times before and by another backend developer

in the end of the project to provide additional human resources to finish the project in-time. The last mentioned developer was slightly more experienced than the core team. As technology changed, the experience of the core team at the beginning of this project is comparable to the experience they have had at the beginning of project B. In both cases, one of both had already little experience with the technology used, and the other had not. What changed is, that also three other colleagues contributed small parts to the project. In project B the members of the core team were the only persons who contributed to the project. This setup will provide more reasonable evidence to the corrective actions of the second iteration as well as the corrective actions of the first iteration because the requirements engineering and code review processes were not changed.

3.2.6 Corrective Actions

The goal of this iteration is a schedule estimation accuracy of 1. Which actions are chosen, why they are chosen and in which way they should help, is described in the following.

In the preceding analysis two causes for the delay of project acceptance were found:

1. Poor resource planning (the team was not finished with working on features on the due day)
2. Long lead and cycle times of fixes and change requests

The resource planning will not be discussed in this master thesis in more detail. Of course, the information, that too little human resources were supplied in project B, will have an impact on the resource planning of project C. The corrective actions of this iteration will focus on reducing the lead and cycle times of fixes and change requests.

Choosing approach

The approach to choose is continuous delivery. Humble and Farley state, that the main benefit of continuous delivery is ”a release process that is repeatable, reliable, and predictable, which in turn generates large reductions in cycle time, and hence gets features and bugfixes to users fast” [39]. The delivery strategy is also part of the six critical success factors in agile software projects described by Chow and Cao [20]. Also Chen and Power report, that they moved 20 applications to continuous delivery. They describe six main benefits [18]:

1. Accelerated time to market
2. Building the right product
3. Improved productivity and efficiency
4. Reliable releases
5. Improved product quality
6. Improved customer satisfaction

The results they have obtained are tremendous. They released every one to six months, now once a week on average and often even multiple times a day when necessary. The release process has become more reliable, caused by the high number of releases. Until the first deployment to production, the fully automated pipeline is tested very often. Therefore the incidents in production have decreased significantly. The software engineers reported that the stress of deploying new code decreased. Finally, the work distribution of the teams is comparable to project B. Approximately 30 % of the effort was fixing bugs. After implementing continuous delivery, the number of open bugs for the applications has decreased by more than 90 %. They also state, that usually nobody is working on customer found failures anymore.

These results are exactly what the following project needs to get accepted on time.

Concrete realization

First of all, the approach chosen is continuous delivery. Continuous delivery provides completely automated a potentially releasable software artifact each time the continuous delivery pipeline is triggered. These artifacts can be deployed to

any environment. In the best case with just one click (one-click-deployments). A step further would be continuous deployment. In this case, the deployment would also be fully automated, and each time the pipeline is triggered, a new version is deployed to production (zero-click-deployments). With the present types of projects, continuous deployment is not desirable.

Continuous delivery is a practice, whose concrete realization differs from company to company and from project to project. Stahl and Bosch described how continuous integration practices differ in the industry of software development [83]. The concrete realization is as close as possible to the recommendations of Humble and Farley [39]. Of course, some parts differ.

Automate almost everything To ensure that the release process gets reliable, it has to be automated. In the concrete implementation, the build is automated, and the test suite is executed automatically.

Develop on mainline Humble and Farley describe developing on mainline the only pattern for branching and merging which enables one to perform continuous integration. This contradicts with the process introduced in iteration 1. As the code reviews and the branching strategy (feature branches) worked well, this strategy will remain. The main reason to develop on the mainline is to avoid ”merge and integration hell” at the end of a project. As the team and the features are small, and the team integrates a feature into mainline as soon as the feature is finished, the present branching approach should not raise problems.

Build the final artifact only once If the final artifact is built only once, one can ensure, that this artifact is reliable, because the automated test suite tested it, deployed to and manually tested on internal quality assurance and customer staging environment, which are as close as possible to the production environment.

Every check-in leads to a potential release This practice gets more complicated as not every check-in is made to the mainline. To not take away the immediate feedback of the automated test suite by developing on branches, the pipeline is also triggered, when commits are made to feature branches. Of course, the created

artifacts are not the same as those on the mainline. Therefore the manual quality assurance does only test features when they are integrated into mainline. Otherwise, the advantage of a single artifact would disappear. To avoid misunderstandings of functionality, the product owner does a quick functional check of every feature on a feature branch before the code gets reviewed and integrated into the mainline. The feature gets tested in more detail from the internal quality assurance, once a release should happen.

Therefore a new state has to be introduced in the process elaborated in iteration 1 (see figure 3.6). The adjusted process is shown in figure 3.17. The adjustments are presented in black. If a developer finishes the coding, the feature is not immediately passed to the technical review anymore but passed to the product owner (functional review). Therefore the latest artifact of every feature branch is deployed to a feature branch environment automatically. On this environment, the functional review is conducted. As this artifact differs from the artifact released later, this functional review is very superficial. If the feature passes the functional review, it is passed to the technical review. Otherwise, the developer gets the feature back to implement necessary changes.

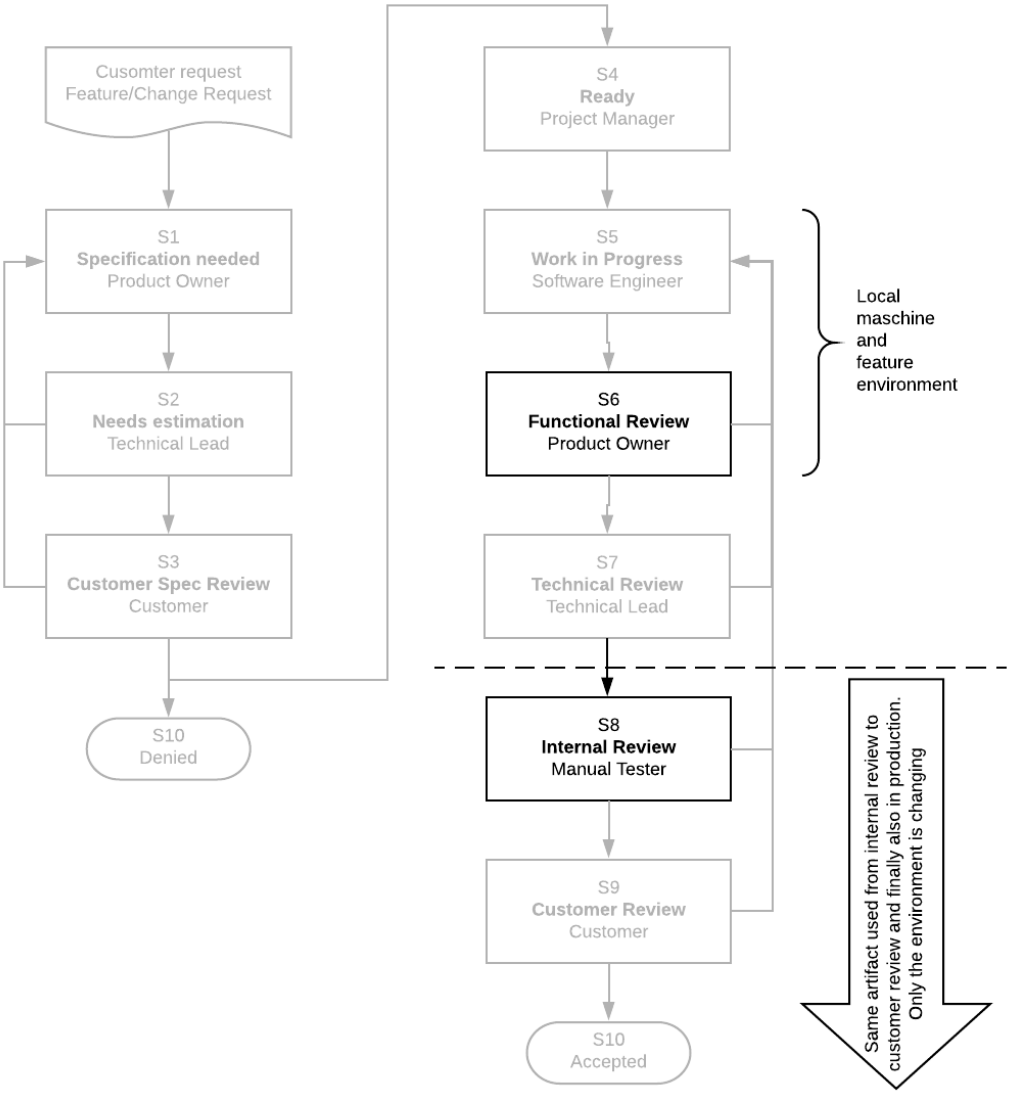

Figure 3.17: Process adjustments for iteration 2 (Changes presented in black)

The role of the manual tester also changes a bit, because the deployment to the internal quality assurance environment and the customers staging environment is now part of the work. This is easy, because these are one-click-deployments. Depending on the situation, changes can be released very frequently or not. This

decision is the responsibility of the product owner. An urgent fix can be reviewed and deployed immediately. This would cause higher manual testing effort, because the system needs to get tested for every change, to ensure that an artifact has a proper quality. But the system is flexible, because the manual tester can also wait for more new functionality integrated in mainline, to test and release them at once. This is of course common practice in the beginning of the project. In addition to that, the manual testing effort will be much less work, because the automated test suite should catch regression defects beforehand.

3.2.7 Analyze Impact on Project C

The analysis is again provided for each goal separately in the following. Tables 3.8, 3.9 and 3.10 contain the data of the three different goals.

Goal 1: Prevent scope creep

By looking at table 3.8, it is clear that scope creep was prevented. Although the project took 28 percent longer than expected (M2) and the total effort exceeded the budget by 40 percent (M4), the project does not show the typical signs of scope creep. In comparison to that, project A had to deal with scope creep by taking twice as long as agreed and by requiring three times more work than estimated. In addition to that, metric M2 is formally higher than informally. This is due to the situation that the project was finished a few days after due day and nothing had to be done anymore, but the customer was on holiday for a week. This delayed acceptance. This is especially important when discussing goal G3, reaching a schedule estimation accuracy of 1.

Questions Q1.3 to Q1.5 are therefore interesting because one can see quickly, that the project was very healthy. In contrast to project B, the effort working on features increased furthermore, whereas the effort working on fixes and change requests slightly decreased. Metric M10 shows that in project C the effort was slightly underestimated, which therefore is the reason for the effort estimation accuracy of 1.4 in this project. If scope creep would not be prevented, the amount of time working on change requests and fixes would have been much higher as it is the case for project A. As nearly no change request effort occurred, it is no

problem, that only 38 percent effort caused by change request was paid. The reason for that is that the project was financially successful and the project manager, therefore, did not even ask for getting all change requests paid. The benefits of some additional hours paid would have less value for the company than the higher customer satisfaction gained by not billing each little change request.

Question Q1.6 shows that the number of failures decreased significantly (M13). The customer found failure rate (M14) seems to be very high, but the point of it is, that all six defects were found within six days and no defect was found afterward. The used definition of the period of observation (first occurrence of a failure to last occurrence) and the actual value of 30 failures found per month is therefore misleading. However, the number of total failures found shows, that the quality of project C is much better than the quality of its predecessors. Therefore trust was successfully built.

Goal 2: Delivering maintainable code

As project B was an extension of project A, the success of code reviews could not be determined clearly. Whereas the size of the code base and the total complexity rose, at least the complexity grew slower than the lines of code. The values of project C are now able to clarify the success of the code reviews introduced in iteration 1.

Table 3.8: G1 prevent scope creep: Measurement results of projects A, B and C

ID	Description	A	B	C
Q1.1	Was the project delivered in-time?			
M1	Estimated Project Duration [days]	64	32	60
M2	Schedule Estimation Accuracy	2.06	1.59	1.28
Q1.2	Was the project built within budget?			
M3	Estimated Project Effort [hours]	243	202	156
M4	Effort Estimation Accuracy	2.79	1.15	1.40
Q1.3	How were resources used?			
M5	Actual Project Effort [hours]	679	233	218
M6	Feature Effort Proportion	55 %	76 %	85 %
M7	Change Request Effort Proportion	37 %	10 %	5 %
M8	Failure Effort Proportion	8 %	15 %	10 %
Q1.4	How were resources used?			
M3	Estimated Project Effort [hours]	243	202	156
M9	Feature and Failure Effort to Estimation Ratio	1.55	0.87	1.19
M10	Feature Effort to Estimation Ratio	1.77	1.04	1.33
Q1.5	Were the change requests paid by the customer?			
M11	Billed Change Request Effort Proportion	3.5 %	86 %	38 %
Q1.6	Was the team successful in building trust by providing a reliable software in time to put the project manager in a good change requests negotiating position?			
M12	Period of Observation [days]	173	213	6
M13	Total Failures Found	112	44	6
M14	Customer Found Failure Rate	19.4	6.2	30
M2	Schedule Estimation Accuracy	2.06	1.59	1.28

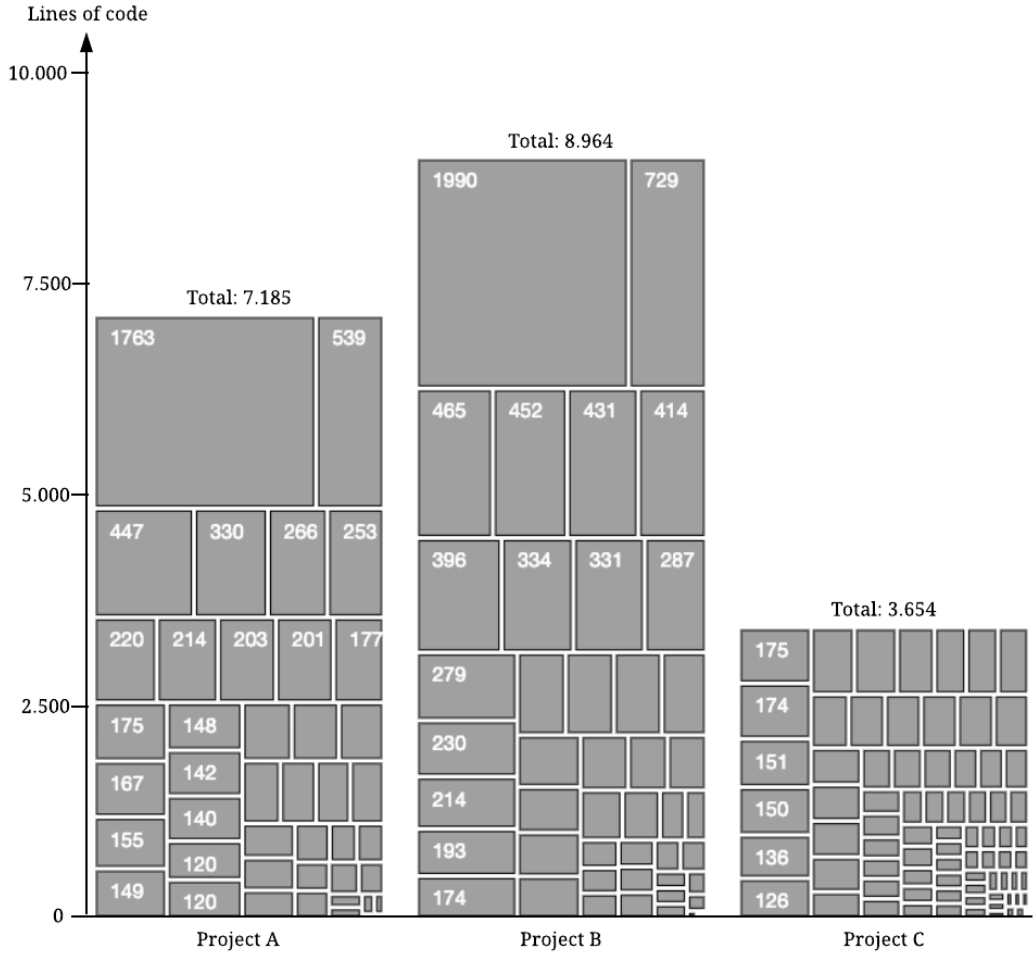

Figure 3.18: Tree map of lines of code of projects A, B and C

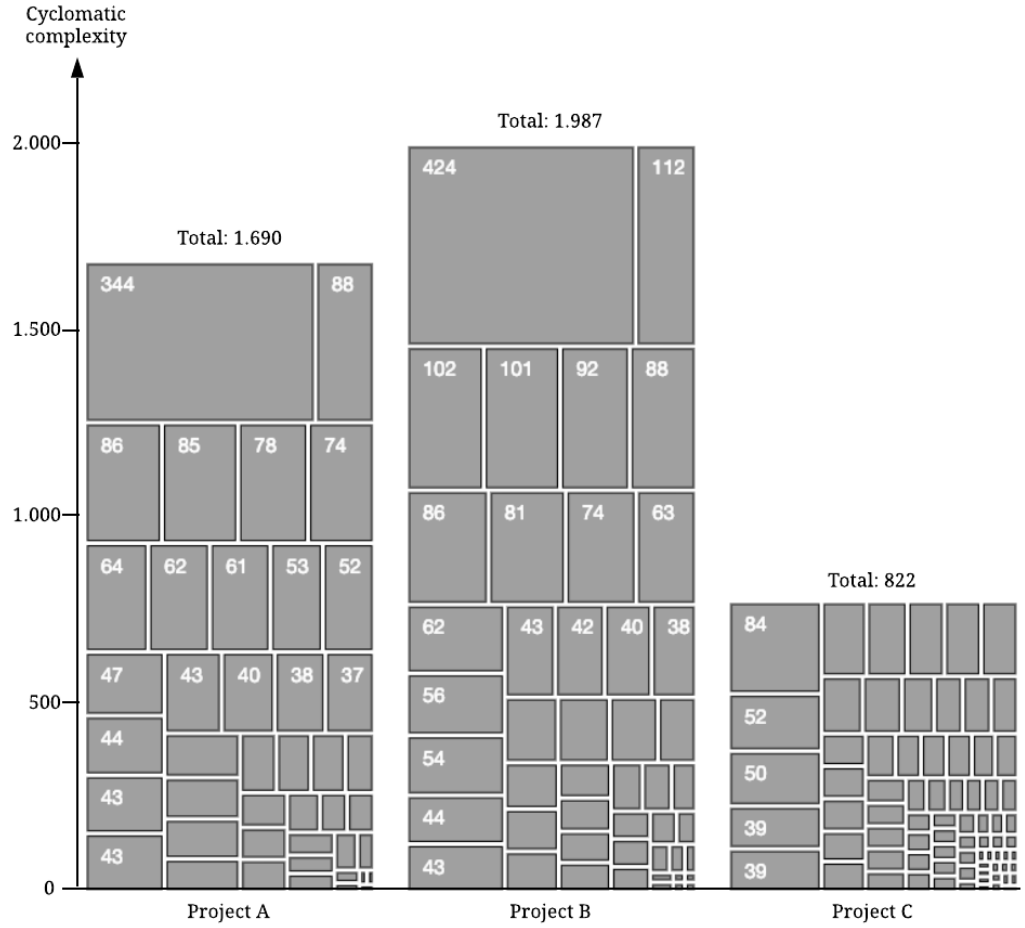

Figure 3.19: Tree map of cyclomatic complexity of projects A, B, and C

As figures 3.18 and 3.19 show, the complexity of the code decreased. This has two reasons:

- Smaller code base
- Better code distribution on files

The code base is 59 percent smaller than the code base of project B, whereas the total complexity decreased only by 58 percent. That means that the complexity even slightly increased concerning the lines of code. Nevertheless, the code is less

complex not only because of the smaller code base. What changed significantly, is the distribution of the code on different files. The lines of code per file and the cyclomatic complexity per file decreased by approximately three quarters (M17 and M20). In figures 3.18 and 3.19 one will notice immediately, that the average lines of code and the average complexity per file decreased and most importantly, that there is no monolith present in project C anymore.

As also an appropriate test suite is supplied in project C and the percentage of duplicated lines is only 1.5 percent, the code is much easier to change and to understand.

In addition to that, the amount of technical debt decreased significantly. Whereas 94 issues per 1000 lines of code were present in project B with an estimated fix time of 141 hours, project C performed much better with only 2.5 issues per 1000 lines of code and an estimated fix time of only two hours.

By looking at all of these values, it gets clear that a well maintainable code was delivered and that the goal is therefore reached. Modern code reviews had a positive impact on maintainability.

Goal 3: Achieving a schedule estimation accuracy of 1

The goal was to deliver on time and get the project accepted on the due day. Therefore looking at metric 2, the schedule estimation accuracy, should suffice. At first glance, a value of 1.28 indicates, that the goal was not reached. However, this value does not represent the true story appropriately. As already mentioned, the customer's person in charge went on holidays some days after the due day. The formal acceptance was then given after customer's holidays. The last features and the last staging defects found by the customer were finished respectively fixed a few days before the due day. The metrics of question Q3.4 show that clearly. Only two percent of the total effort was done after the due day. The product was already finished on the due day, and the acceptance was only delayed by the customer's holidays. Ultimately the customer was satisfied. Therefore the schedule estimation accuracy of 1.4 does not matter, and the goal was reached. In the following a discussion about the impact of continuous delivery and the reasons for reaching the goal is provided.

Table 3.9: G2 deliver maintainable code: Measurement results of projects A, B and C

ID	Description	A	B	C
Q2.1	How complex is the code?			
M15	Lines of Code	7185	8964	3654
M16	Maximum Logical Lines of Code	1800	1990	175
M17	Average Logical Lines of Code	167.1	190.7	47.5
M18	Cyclomatic Complexity	1690	1987	822
M19	Maximum Cyclomatic Complexity	344	424	84
M20	Average Cyclomatic Complexity	39.3	42.3	10.7
Q2.2	Is the code well documented by a meaningful test suite?			
M21	Appropriate automated test suite applied	No	No	Yes
Q2.3	How hard is it to change the code?			
M22	Percentage of Duplicated Lines	6.8 %	2.6 %	1.5 %
M21	Appropriate automated test suite applied	No	No	Yes
Q2.4	Is the amount of technical debt acceptable?			
M23	SonarQube Issues per 1000 Lines of Code	140	94	2.5
M24	SonarQube Fix Time [hours]	164	141	2
M22	Percentage of Duplicated Lines	6,8 %	2.6 %	1.5 %

In project C the resource planning was better than in project B. Project C required 40 percent more resources (M4) than expected and the project was nevertheless delivered on time. This shows that enough human resources were provided and also enough safety buffer was taken into account. In project B the team was working on features until the due day and also after it. In contrast to that, in project C the team was already working on fixes and change requests before the due day (question Q3.6). This indicates a good schedule because after the due day the team worked just on some little change requests (M32). The features were already finished, and the system was stable.

Reaching goal G3 could be traced back to the fact that resource planning was improved. However, question Q3.7 shows, that continuous delivery had a huge positive impact on the results. The given metrics, especially the cycle times are key performance indicators for the continuous delivery system. Figure 3.20 shows that cycle and lead times improved significantly in project C. The median lead time decreased by 61 percent and the median cycle time by 49 percent. In addition to better median values, the lead and cycle time is more reliable in project C. The interquartile ranges of lead and cycle times decreased by 65 percent respectively 60 percent. In addition to that, there are no outliers present in project C. As mentioned in the analysis of project B, these values have a big impact on in-time delivery. As project B's planned duration was 32 days, a median lead time of more than seven days for fixes and change requests is a long time, especially after due day, when the customer reports defects and the acceptance is blocked until they are fixed. In contrast to that, project C had a planned duration of 60 days and a median lead time of three days. By only looking at the defects, the median lead times are still nearly seven days in project B, and only one day in project C. This is a crucial factor when the customer starts to test the system and the due day is near. Even if a high amount of defects is detected, the team can fix those immediately, and the project can still be accepted as planned. In project B, this was not possible.

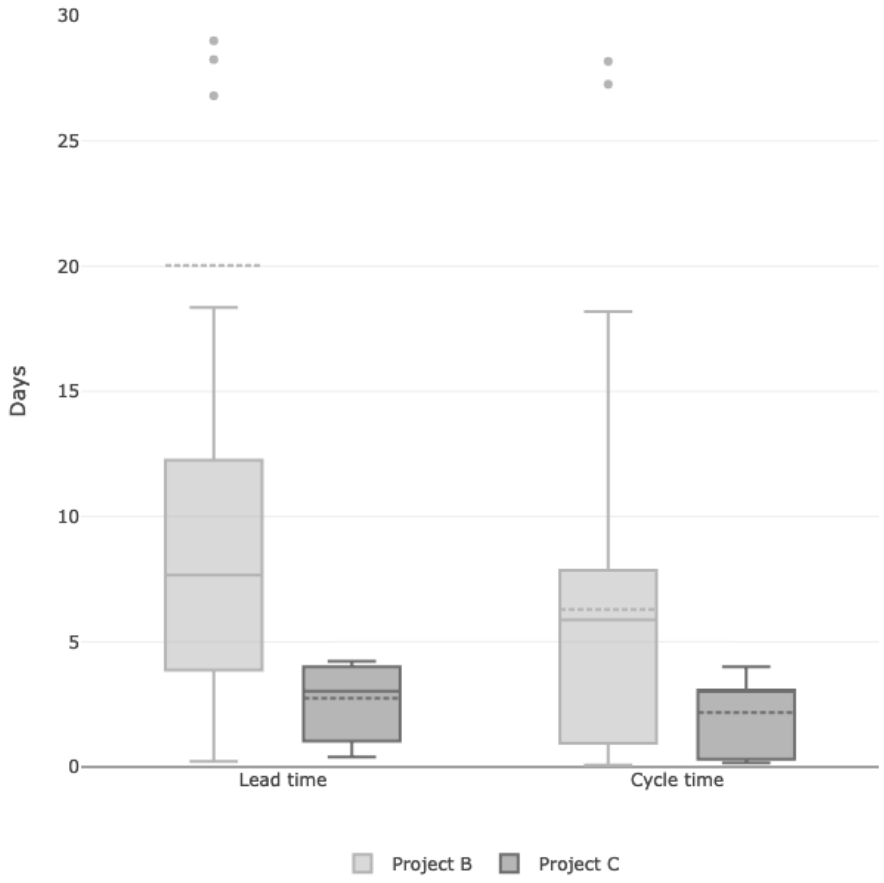

Figure 3.20: Lead and cycle times of projects B and C

In addition to the low lead and cycle times, also the number of failures found by the customer decreased. Therefore the results of nearly no defects and significantly lower lead and cycle times reported by Chen and Power [18] could be reproduced in project C. The positive impact of continuous delivery could be proved. The cycle and lead times decreased on average and became more reliable too.

Table 3.10: G3 achieve a schedule estimation accuracy of 1: Results of projects A, B and C

ID	Description	A	B	C
Q3.1	Was the project delivered in-time?			
M1	Estimated Project Duration [days]	64	32	60
M2	Schedule Estimation Accuracy	2.06	1.59	1.28
Q3.2	Did a wrong effort estimation caused the delay of acceptance?			
M3	Estimated Project Effort [hours]	243	202	156
M4	Effort Estimation Accuracy	2.79	1.15	1.40
Q3.3	Was the planning of resources appropriate?			
M3	Estimated Project Effort [hours]	243	202	156
M25	Invested hours by estimated hours until due day	1.57	0.79	1.37
Q3.4	How much of the work was done from due day to day of acceptance?			
M26	Total hours spent until due day	381	160	213
M27	Total hours spent since due day to day of acceptance	274	33.5	4.25
M28	Invested hours by estimated hours until due day	42 %	17 %	2 %
Q3.5	How was the time used from due day to day of acceptance?			
M29	Prop. of hours worked on features since due day	43 %	40 %	0 %
M30	Prop. of hours worked on failures found by customer since due day	15 %	16 %	0 %
M31	Prop. of hours worked on failures found by QA since due day	-	-	0 %
M32	Prop. of hours worked on change requests since due day	43 %	44 %	100 %
Q3.6	How was the time used until due day?			
M33	Prop. of hours worked on features until due day	66 %	99 %	87 %

M34	Prop. of hours worked on failures found by customer until due day	1.7 %	0.2 %	3.4 %
M35	Prop. of hours worked on failures found by QA until due day	-	-	7 %
M36	Prop. of hours worked on change requests until due day	32 %	0.6 %	2.6 %
Q3.7	How easy was it to deliver change requests and fixes?			
M37	Median lead time for fixes and change requests	-	7.66	3.02
M38	IQR of lead times for fixes and change requests	-	8.38	2.97
M39	Median cycle time for fixes and change requests	-	5.87	3.01
M40	IQR of cycle times for fixes and change requests	-	6.91	2.75

Threats to validity

As project B was an extension of the product developed in project A, section 3.1.8 raised some doubt. It was concluded that well-defined requirements produce the desired results, but it was not clear if it is possible to generate such well-defined requirements also in the context of a new product, where the business domain is not known that well. This also limited the expressiveness of the results obtained by modern code reviews. As project C is a new product (see description of project C in section 3.2.5), the second iteration allows not only to validate continuous delivery, but also the requirements engineering and modern code review processes applied in the first iteration.

As the results of project C show, scope creep was also prevented in an unknown domain. The results of iteration 1 can, therefore, be confirmed. However, it has to be mentioned, that the estimation was not as accurate as in the well-known domain of project B. Project B needed 15 percent more resources than expected and project C 40 percent more. However, it has to be said, that both results are sufficient and can be seen as a success. Scope creep only occurred in project A, which caused 179 percent more effort than expected. An additional indicator is the change request effort proportion. Whereas in project A 37 percent of the overall effort was used for working on mainly unpaid change requests, in project B and

C this proportions shrank to 10 respectively 5 percent, which were also mainly billable. The hypothesis that an appropriate requirements engineering process prevents a project from scope creep can be confirmed.

Regarding the goal of delivering maintainable code, project C showed significant improvements to its predecessors. The code quality of project A was bad. This was mainly due to the wrong usage of the Concrete 5 framework by an inexperienced developer and by changing the code frequently due to the high number of change requests. The requirements engineering process already prevented project C from this high number of change requests. This could already be one reason, why the quality of the code was better in project C. The intention of the modern code reviews was the avoidance of large design or usage mistakes as made in project A. This was achieved in project C. However, also this could have another reason. As mentioned, the skeleton of the code was set up by developers, which were already familiar with the technologies used. The senior developer performing the modern code reviews had no reason to intervene. Therefore the significant better code quality was at least not only caused by the application of modern code reviews. However, it has to be said, that a scenario, as occurred in project A, would have been avoided surely, because such wrong usages would have been found. In addition to that, the inexperienced developers felt qualitatively more comfortable by knowing, that a senior developer will review the code. Also many small optimizations, yet not known by the inexperienced developers were provided. The project was, therefore, better monitored with just a little effort from a senior developer. These reviews could, therefore, be recommended for very small enterprises.

The involved personnel can qualitatively confirm the positive impact of continuous delivery on the schedule estimation accuracy and on the lead and cycle times. In situations with high pressure regarding time, the automated delivery including an appropriate test suite prevented the team from delivering error-prone code to the customer many times. This accelerated the delivery of new functionality to the customer significantly. The author of this work was the project manager of all projects and also acted as a developer. He is convinced, that project C would have exceeded the deadline drastically if continuous delivery would not have been applied to the project. His conviction is based on the many failures, which were introduced and not delivered to the customer. In addition to that, new functionality was developed, and changes were introduced more rapidly. This is caused by the

improved confidence provided by the test-suite. The results obtained can, therefore, be qualitatively confirmed.

4 Related Work

The first part of this chapter presents a classification framework for software process improvement. This classification framework is created in the style of an existing framework combined with useful adjustments for classifying this work in particular. This framework helps to get a good overview of topics, which are part of this work, and to other related fields of those topics. The localization of this work is also provided in the first part. The second part of this chapter describes, what kind of publications are treated as related work. In the third part, the related publications are presented according to the criteria elaborated in the second part.

4.1 Classification Framework

The created classification framework can be found in figure 4.1. First of all, the master thesis is about enabling project success in very small companies. In section 2.2.2 the different categories of success factors of agile projects are presented. These factors are derived from Chow and Cao [20]. In addition to that, one can also find the reasons, why this thesis focuses on process and technical factors. Of course, the size of the company under investigation is also an important attribute [41]. Therefore, related work is classified in publications, which target very small enterprises, small and medium enterprises, and large enterprises. The definition of the concrete number of employees of each category differs. Truly related are publications, which target companies with less than ten full-time employees. This is because companies with less than ten full-time employees strongly differ from other small companies with 10 to 50 full-time employees [36]. Also, the application domain and the region should be taken into account. The investigated company is located in Austria and focuses on web and mobile applications.

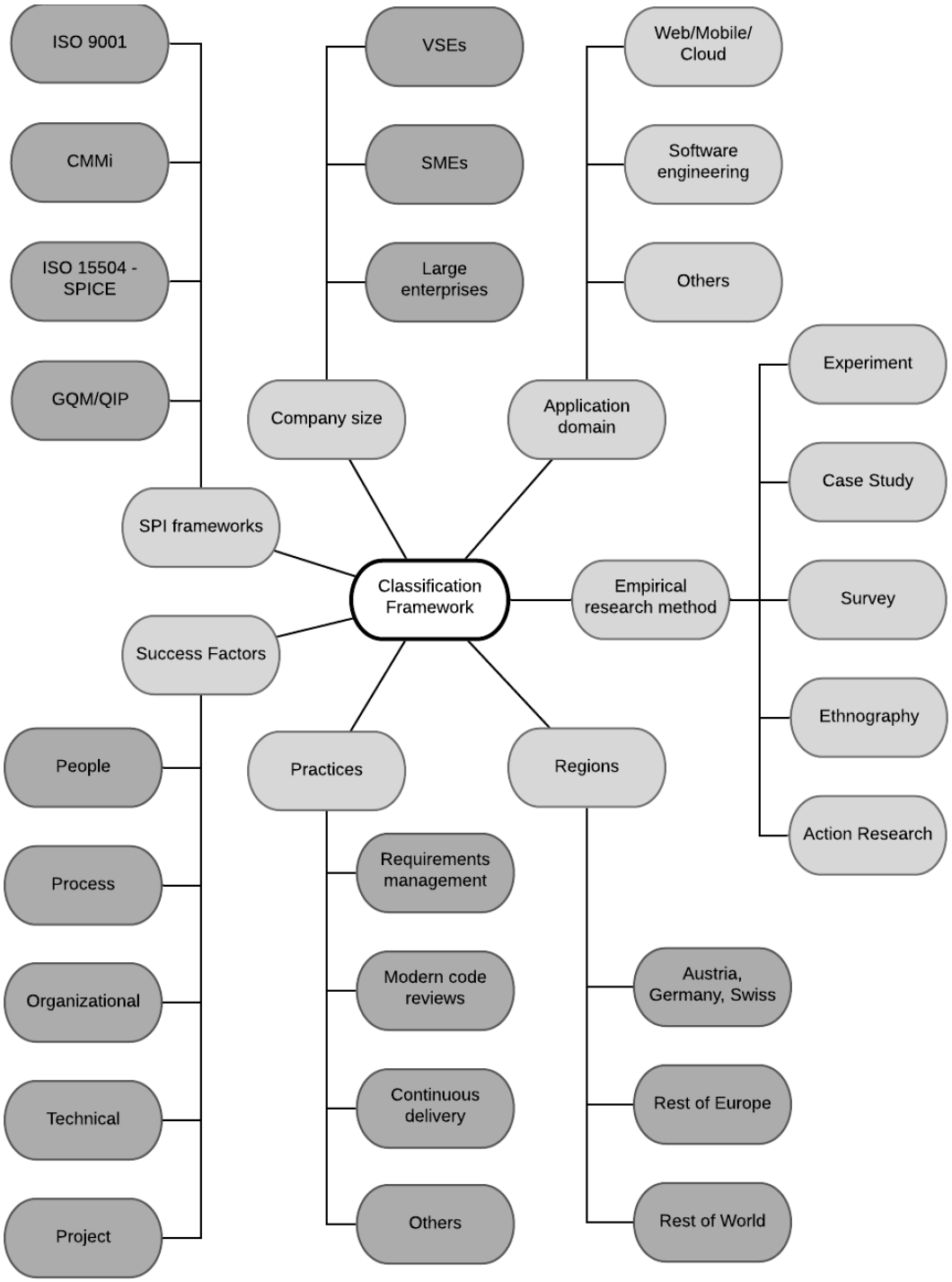

Figure 4.1: Classification framework

The attributes mentioned so far refer to the context of the organization. The SPI frameworks, the research method, and the practices describe the methodology. This thesis is an empirical action study. The categorization of the empirical methods is derived from Easterbrook et al. [25]. Also, software process improvement can be made in many different ways. Often standards or recommended systematic approaches, like CMMI, ISO 9001, SPICE or GQM/QIP, are used. A detailed discussion about these frameworks is provided in section 2.3.1. In addition to the concrete framework, the different software engineering practices are used in the classification framework. As there are too many practices to be illustrated in this classification framework, e.g., the twelve extreme programming practices [10], only requirements management, modern code reviews and continuous delivery, the practices used in this work, are illustrated.

4.2 Inclusions and Exclusions

As the topic is broad, it is important to find an appropriate definition of related work. Of course, there are no publications, which would be classified the same way as this work. Therefore the classification framework is too detailed. In the following, the definition of related work is elaborated.

As a GQM-model is used to quantify all observations, one key aspect for related work is an empirical quantitative research method, e.g., case studies, controlled experiments or action researches. The quantitative nature is important for comparing the results.

Case studies and action researches about small companies executing a SPI initiative with quantitative results are therefore related. A restriction for the region, the different practices and using exactly the practices of this work is not useful but would be highlighted if one of those attributes is also fulfilled.

Regarding the SPI framework, publications, which used a GQM model in an iterative way like QIP, are treated as related work. Publications regarding other SPI frameworks are only considered as related if the region and some practices intersect with this thesis.

So far, only publications are considered as related work, which have the same context and most importantly the same or a similar overall methodology. However, this study also shows the impact of the different practices on project success and on other software engineering topics independently. Therefore empirical quantitative researches, which focus only on the impact of one or more of the used practices on project success can be treated as related work.

4.3 Publications

In the following, publications were presented, which fulfill the criteria elaborated before.

Software process improvement is topic of interest for more than two decades now. Around the year of 2005 research concerning software process improvement of small enterprises gained interest. As an example, Von Wangenheim et al. reported their experiences in establishing software processes in two small software companies [91]. They were using the ASPE-MSC approach for the establishment of the software processes. This approach is similar to the GQM/QIP approach. It is an iterative approach, which identifies the high-priority process areas at the beginning of each iteration. These processes are then defined, implemented and ultimately evaluated. The first company had 19 full-time and 3 part-time employees. For this company, they describe the implementation of a change request process, as this work did in iteration 1. They observed mostly qualitative benefits. The time spent on the handling of change requests decreased. The monitoring and traceability of change requests inside the company have improved significantly. They observed a reduction of 70 percent in the time spent on searching for information on specific requests. The estimation worked better, cases of cost overruns and late implementations were reduced, as in this master thesis. In addition to that, they observed an increased customer satisfaction. The second company had five full-time and five part-time employees, which is nearly the same setting as it is in the investigated company of this work. The priority was on establishing supply, customer support, and software development processes. Only two percent of the productive capacity of the company was designated to the process establishment. This is reasonable when considering the benefits. Due to automation, the software

process matured and was applied more consistently across projects and individuals. The number of service-packs for fixing defects after delivery halved. Ultimately the reduction of delays by incomplete execution of process steps was observed, e.g., not all required pre-conditions were checked in advance of the installation process. This publication is exemplary. The study is also confirming other experiences [8][48][59][79][78] indicating, that it is possible to define and implement processes also in the context of small companies in a beneficial and cost-efficient manner. As this master thesis also shows, software process improvement is generating measurable improvement and advances also in very small companies.

Clarke and O'Connor also investigated the influence of software process improvement in SMEs [23]. In contrast to many other studies including this master thesis, they did not focus on increases in productivity, product quality and customer satisfaction, improvements to budget and schedule adherence and decreases in costs, cycle times and process complexity. They directly analyzed the business success of those small companies. They found that business success was positively correlated to the amount of software process improvement activities. The 15 investigated organizations had a headcount between 4 and 120, whereas 7 of them had one of less than 20. Many others did this by calculating the return on investment (ROI). As an example, Van Solingen collected a bunch of ROI calculations of different big companies like Motorola and Boeing - determining that the average ROI for SPI is 7:1 [89]. It has to be said, that ROI is viewed skeptically due to inconsistent calculations [27].

Another example explained in detail calculated the reduction in percent of the total effort per one level change in a five-level-process-maturity-scale [22]. They found that the larger a project regarding the lines of code is, the more savings are achieved. This study was applied on 161 projects from 18 sources within 2,600 and 1,264,000 lines of code. As one can see, small projects are only a minor part of it. However, there is one interesting thing indicating a chance for small companies with small projects. The effort reduction grows logarithmically with the size of the lines of code. This means that on the one hand, the relative effort savings grow with the size of the project indicating fewer advantages for small projects. On the other hand, the relative effort savings per one level change in process maturity grow faster within smaller projects. Therefore, from 200,000 lines of code to 250,000 lines of code, the effort reduction raises from approximately 8.7 percent to 9.1 percent. That means 4.5 percent more relative effort savings per one level change in

process maturity. In small settings, this is significantly different. The effort savings from 2,600 to 50,000 lines of code start by approximately 3.8 percent and end by 6.7 percent, resulting in an increase of 66 percent in efficiency. This means that companies with small projects do not gain many benefits by increasing the process maturity. However, at the moment they are starting to grow and to get bigger projects, a one level change in process maturity would have had a big positive impact. One can see, that process maturity is important for scaling the business.

Harter et al. investigated the contradiction of high quality and low cycle times respectively less effort [35]. Contrary to common belief, they found, that improvements in both aspects can be achieved simultaneously. They found, by examining 30 software products by a major IT firm, that higher maturity levels are associated with higher product quality, but also with increases in the development effort. However, their findings indicate, that the reductions in cycle time and effort due to improved quality outweigh the increases from achieving higher levels of process maturity. This master thesis also increased maturity. The findings of Harter et al. can be confirmed. The quality rose and the cycle times significantly decreased.

A comprehensive study by Galin and Avrahami analyzed 19 papers with results of more than 400 projects from organizations in several countries, which adopted CMM [31]. The values are given in relative improvements in order to a single level advance in CMM. They found that the median error density (the number of errors per lines of code) decreased by 26 to 63 percent. They also found a median decrease of 34 to 40 percent of rework effort. In addition to that, the median cycle time of projects decreased in a rage from 28 to 53 percent. The companies compared the duration of similar projects before the advance of the CMM level. These findings correlate with the results of this master thesis, although it targeted large companies and CMM as SPI framework.

Regarding the severity of defects, Harter et al. found out, that a higher level of process maturity decreased the likelihood of severe defects [34]. They also found that high complexity projects benefit more from a higher maturity that simple projects do. This correlates with the findings of B. K. Clark already described above [22].

The previous studies show in general, that software process improvement initiatives in general and in small companies are worth the effort. Many of them show

qualitative benefits by applying software process improvement. The remaining studies evaluated the impact on different quality indicators like the number of defects, the amount of rework and customer satisfaction. Generally spoken, the presented publications show the impact of process maturity on companies, especially on small companies. In contrast to that, the following publications show the impact of the different corrective actions applied in this master thesis, namely requirements engineering, modern code reviews, and continuous delivery. In this case, the application of these practices does not have to happen within a software process improvement initiative. Although this master thesis' main approach is software process improvement and its empirical quantitative evaluation with a GQM-model, these studies are also relevant. This is because the iterative approach also allows a comparison of the different quantitative results of the corrective actions separately.

Requirements engineering is the first practice to be investigated. First of all, publications are investigated, which analyzed the impact of a requirements engineering and change management process, preferably on small companies. The case study applied by von Wangenheim et al., already described above, shows the impact of a change request process including a sufficient description of change requests [91]. As mentioned, the time spent on the handling of change requests decreased. The monitoring and traceability of change requests inside the company have improved significantly. They observed a reduction of 70 percent in the time spent on searching for information on specific requests. The estimation worked better, cases of cost overruns and late implementations were reduced.

Many publications claim that requirements engineering is positively associated with improved productivity, software quality, and risk management. As an example, Erik Simmons stated in his presentation about his lessons learned in five years of requirements engineering improvement at Intel, that the ROI of requirements engineering is large enough that its absolute value is irrelevant [80]. This statement is representative for many others because there is little evidence to support these statements about huge benefits [24].

Other studies, on the contrary, show the negative effects of not using appropriate requirements engineering. Quispe et al. interviewed 24 project managers from 24 different very small enterprises with fewer than ten developers. Their findings indi-

cate that the specifications are usually met, but the client often finds the solution unsatisfactory, most often caused by communication issues with clients resulting in incomplete specifications. In addition to that, the project's scope expands as clients require additional changes. As they also found that very small enterprises use an ad-hoc process, issues like loss of requirements arise and developers tend to resolve the issue without contacting the clients [68].

Aranda et al. were investigating the requirements engineering practices of seven small companies [3]. Three of them had less than ten employees and the rest between 19 and 45. They identified four major findings, which they used to formulate hypotheses for further investigation. On the one hand, they found, that everyone is doing requirements engineering differently. On the other hand, they found three things the companies had in common. The companies had a high cultural cohesion, the CEO was in four of seven cases the requirements engineer, as in the company of this master thesis, and requirements errors did not cause catastrophes for the investigated companies so far.

To sum up, on the one hand, literature is out there, which described the problems caused by poor requirements engineering and which practices companies use to overcome those. On the other hand, only a few publications provide evidence for the claimed positive effects of requirements engineering. This master thesis confirms the problems stated above and quantifies the positive impact of an appropriate requirements engineering and change request management process. This should increase the trust of practitioners, that requirements engineering pays off.

Modern code reviews and its impact is investigated recently. Bird et al. found out, that the most comments of modern code reviews relate to code improvements and knowledge transfer as it is supporting understanding and social communication. It is mainly used to ensure the code's long-term maintainability, as a knowledge sharing tool and to broadcast ongoing progress. These things are expected, and modern code reviews fulfill that. Another main expectation is to find defects, but effectively little defects are found [6]. This is similar to the findings of two other publications, which found a ratio of 75:25 of maintainability-related changes to functional changes caused by modern code reviews [11][52]. These findings reflect the intention and the subjective perception of the people involved in this master thesis and the projects investigated.

The publications presented so far, primarily show the content of the comments and the changes made by modern code reviews. In addition to that, some publications investigated the impact of different parameters, e.g., review coverage, of modern code reviews on software quality. McIntosh et al. studied the relationship between post-release defects and code review coverage, code review participation and the domain-specific code reviewer expertise. They found that these parameters share a significant link with software quality [57][58].

Continuous delivery is the third and last practice applied in the software process improvement initiative of this master thesis. The practice is quite new, but there are already many publications, that show the positive impact of continuous delivery. Chen reports huge benefits in moving 20 applications to continuous delivery at Paddy Power, a rapidly growing company with 4000 employees. The release frequency has increased from once every one to six months to application releases once a week on average, even multiple times a day when necessary. The cycle time from conception of a user story to the deployment to production decreased from several months to two to five days. The time to market therefore accelerated. The product quality improved additionally. The number of open bugs decreased by more than 90 percent. He also reported a lower level of stress for the development and operations team [19]. The impact was also analyzed in the B2B domain. The findings are pretty much the same. Increasing speed, quality, and capacity of development is reported [70]. These improvements can be confirmed quantitatively by this master thesis. Project C had significant lower lead and cycle times along with a decreasing number of defects found by the customer.

In addition to that, the satisfaction of the team was higher in respect to subjective perception. This perception was empirically analyzed by Kropp et al. They conducted an online survey in Switzerland in 2016 and investigated the impact of agile development itself and its practices on satisfaction of practitioners. They found that a low time to market correlates with high satisfaction. Additionally they found, that the most satisfied practitioners more often use the practices of automated builds, continuous integration and continuous delivery than the most unsatisfied practitioners do [46].

As one can see, there is only little academic literature that provides empirical

analysis of software process improvement, requirements engineering, modern code reviews and continuous delivery in the context of very small enterprises. This could be a reason, why very small enterprises do not start software process improvement initiatives and do not use state of the art software practices. This master thesis provides quantitative results and should therefore weaken the common belief, that such initiatives are only for large companies and do not pay off for very small companies.

5 Conclusions

This action research dealt with project success in very small enterprises. Throughout 14 months, a light-weight software process improvement initiative was conducted to a very small web development company in Austria. The software industry is still one of the fastest growing worldwide. For instance, millions of jobs are provided by software companies in the European Union and the United States. About ninety percent of these companies are very small and have a headcount lower than ten. As a result, these very small companies have a huge economic impact. As these companies are most often cash flow driven, project success is vital to them. Over the years software process improvement was not only evaluated and examined in the context of large companies, but also in the context of small and medium sized enterprises. This helped to break the belief that software process improvement is only applicable in large companies and is accompanied by a huge overhead. Unfortunately, there are big differences between small and very small companies. Consequently, the evidence provided is not fully transferable to very small companies. The goal of this work was to provide the missing evidence that very small companies also benefit from software process improvement initiatives. In the following, a recap of the thesis is provided (see 5.1). In addition to that, discussions about the used software process improvement approach (see 5.2) and the different used practices including threats to validity (see 5.3) are provided.

5.1 Recap

In chapter 2 the investigated company is presented in detail. The company has undergone rapid growth in workforce. The number of employees quadrupled within two years and the total hours worked per month grew exponentially. The company was informally organized as a star around the founder. As the workload and the

number of employees increased, the founder was not able to meet his responsibilities anymore. Requirements, estimations, code quality, and the technical portfolio were important but not urgent responsibilities and were therefore often omitted. Project success in the sense of providing appropriate quality and delivering the right product within time and budget constraints disappeared. Many other very small companies have the same problems.

By examining the five critical success factors for agile projects, namely organizational, people, process, technical and project factors, the process and technical factors were chosen to enable project success in the investigated company. In the following different software process improvement frameworks, like ISO 9001, CMMI, QIP/GQM, and SPICE were presented. The bottom-up approach of QIP/GQM was chosen. This enabled the company to set its own focus areas instead of using predefined areas such as it would be necessary by using CMMI and SPICE. In addition to that, the Quality Improvement Plan is iterative. Combined with the Goal Question Metric approach, no big effort has to be made up front. Further, it avoids unnecessary overhead, which is one of the key anxieties of very small companies. In chapter 3 two iterations are described. Both iterations consist of an initial observation of the preceding project. From these observations, goals for the next project are derived. These goals were used as a basis for the Goal Question Metric approach. With this approach, a quality model was created in the first iteration and extended in the second. After completing the model, it was used to quantify the observations of the preceding project. With this comprehensive analysis of the present problems, appropriate corrective actions were retrieved from literature. These actions were applied in the subsequent project. As the last step, the success of the corrective actions was evaluated by the quality model. As already mentioned, the whole process was executed two times. Three projects A, B and C within fourteen months were therefore included, whereas the first project only served as an observation and analysis basis. Therefore the results of the first project were not affected by instructions of this work.

In the first iteration, project A was analyzed. The observations indicated the presence of scope creep and a lack of quality. Scope creep is a dynamic that is caused by poor requirements engineering causing many change requests, which force the development team to exceed estimated effort and the deadline. As time pressure increases, the number of failures found by the customer also increases. Finally, the trust of the customer disappears. As a result, the customer is not willing to pay for

the additional effort caused by the change requests. Heavily missed deadlines, a widely exceeded budget, and an unsatisfied customer are the consequences. The goals of the first iterations were, therefore, preventing scope creep and delivering maintainable code. The created quality model confirmed the observations. Project A took twice as long as determined and took nearly the triple amount of estimated effort. A third of this effort was caused by change requests, whereas only a tiny proportion of this effort was billed. The analysis of the code showed an enormous technical debt accompanied by the fact, that no test suite was applied. To prevent project B from such a scenario an appropriate requirements engineering process was introduced and modern code reviews were integrated into the software development process. These corrective actions had the desired impact. Project B only took 60 percent longer than expected and was nearly finished within the estimated effort, mainly due to fewer change requests. Also, the number of failures found by the customer decreased by 60 percent. As project B was an extension of the product developed in project A, the impact of modern code reviews on the code base was noticeable, but not groundbreaking. Whereas the number of code smells and the proportion of duplicated lines decreased, most of the bad code base did not improve as code reviews only affected the new code. Additionally, there was still no test suite applied.

In the second iteration, project B was analyzed. In contrast to the analysis of project A in iteration 1, project B could be analyzed not only by subjective observations but also by the quality model created in iteration 1. Based on this analysis, getting the project accepted in-time was the goal of iteration 2. This new goal extended the quality model. The quantitative analysis with the updated quality model of project B showed, that long cycle respectively lead times of fixes and change requests in addition to a poor resource planning caused the delay of acceptance in project B. To shorten the cycle and lead times, the practice of continuous delivery was established in project C. Therefore a comprehensive automated test suite was also applied. The result was very satisfying, although the formal acceptance was late. This happened because the customer's person in charge went for holidays some days after the due day. This was deferring the formal acceptance to the end of the customer's holidays. As the project was informally accepted beforehand, the project was de facto accepted in-time. This was accomplished even though 40 percent more resources were used than expected. This is mainly caused by a decrease of 65 respectively 60 percent of lead and cycle times. This enabled the team to deliver final fixes and change requests faster without introducing new

defects. The number of failures found by the customer decreased by 86 percent. The customer found only six failures in total. The duration from the first review to the final acceptance was therefore very short. In addition to that, the impact of modern code reviews in the software development process became visible. The code base was significantly cleaner than the code base of the product developed in project A and B.

5.2 Discussion of SPI Approach

The first perspective is to look at the software process improvement initiative as a whole. Thus the methodology of elaborating a quality model with the Goal Question Metric approach in combination with choosing, applying and evaluating appropriate corrective actions can be evaluated. In the case of this action research, the approach worked very well. As the quality model is based on specific goals, which one wants to achieve by changing the process, or by establishing the use of different practices, there is no overhead implied. It enables executives to focus only on specific areas. This is a very beneficial factor, because this work shows, that already small and specific changes can have a huge impact. This work provides evidence that no blown up assessment or software process improvement framework is necessary to enable project success in very small companies.

Applying one iteration does not need to be accompanied by a software process improvement initiative. It is also very well suited to introduce goal-driven change, which should be verified quantitatively. The problems in very small companies are most often well known by all parties. The definition of a goal can be done quickly, as large structured assessments are not necessary for such small entities. Also, the elaboration of the questions and metrics does not need a considerable amount of time. It should be possible to define the goal, the questions and the metrics within one day. One should be careful by the selection of the metrics. The most work of this goal-oriented approach is collecting the data after the definition of the GQM-model. By carefully selecting metrics that are expressive but not extremely hard to gather, one can increase the efficiency of this approach significantly. The problem of this approach is that maybe not all metrics can be extracted from the legacy project. If it was not possible to extract the most meaningful metrics from the legacy project, one would have to run another project without introducing

the planned practices to get an object of comparison. In the case of this thesis, this problem did not occur. However, it has to be said, that much manual post-processing was necessary.

By applying this approach more often, so to say extending the GQM-model by new goals for additional changes, the model could get confusing due to a missing structure. This is a disadvantage of the bottom-up nature of this approach. The GQM-model of this thesis was created in the first iteration and extended by the second and is already quite complex. If some iterations are performed, one could think of redesigning the quality model by changing to a top-down approach. Maybe the maturity of the company is then already high enough to implement more formal standards like CMMI, SPICE or ISO 9001. Another possibility could be to decrease the level of detail of goals, which were accomplished many times consecutively. As an example, the first question of the first goal of this work would suffice to determine if the goal is accomplished. Further questions are only needed to find the root causes. In case of success, they are only needed if one wants to verify if the corrective actions impacted the project in the way expected or not. As an example, if one wants to know if the project did not exceed the budget, because the number of change requests was reduced. By applying the corrective action the first time, this can help to verify if the results are caused by the action or by other factors. This is also discussed in section 5.3. As one can see, in later iterations a high level of detail is not needed anymore, and the complexity of older goals can be decreased. Especially, it is no problem to elaborate these detailed metrics in case of failure. Another possibility would be to introduce thresholds, which automatically indicate if a goal is reached or not. This would allow the whole company to see the status of projects all the time. Of course, a quality manager could interpret the model every once in a while to report to executives. However, in such small companies, it is very likely that the executives would not delegate this interpretation.

5.3 Discussion of Applied Practices and Threats to Validity

Little has been said about the limitations and the validity of this work. The evidence provided by this work is only carried by the examination of one company. Of course, this limits the expressiveness of this work. This is especially the case in the context of very small companies because project outcomes of such small companies are strongly affected by individual human contributions. More empirical evidence would be beneficial to confirm the findings of this work. Regarding the validity of the gathered data, it has to be said, that very small companies have limited possibilities to collect data, which is entered by employees, precisely. As the amount of data is small, for example the time logged on features by developers, already a few inaccurate or wrong manual logs can influence the data significantly. Therefore it is essential to assess the validity of the data collected, by comparing the outcomes with the subjective perception of the people involved. In this work, the subjective perception of the people involved correlates strongly with the data collected by the quality model. As already mentioned, further empirical evidence could, therefore, prove the validity of the obtained results.

The threats to validity are discussed in detail at the end of the analyzing chapters of each iteration (see 3.1.8 and 3.2.7). As project B extended the product created in project A and the team did not change and was, therefore, more experienced and familiar with the product in project B, the positive impact of requirements engineering is questionable. As already mentioned before, the foremost questions related to each goal show if the goal was reached or not. In contrast to that, the subsequent questions show, how a goal was reached or why a goal was not reached. It turned out that scope creep was prevented because the requirements were defined very well. The remaining question was if the requirements were defined well because the domain was already known. This doubt could be dispelled in the second iteration. At the beginning of project C, the domain was not known, the project team was not familiar with the technologies used, and another customer ordered the project. The initial setting was therefore comparable with project A. The requirements engineering process performed well also in this situation.

The improvement of maintainability in project B and C could not be solely traced back to modern code reviews, neither in the first nor in the second iteration. As the

requirements engineering avoided a high number of change requests, the code in project B and C was not changed as often as it was changed in project A. Of course, this influences the maintainability of the code positively. There is no doubt that both facts had a positive impact. However, it is not clear how large the individual contributions are. The personnel situation had no impact because the experience level of the teams working on project A and C were comparable. Modern code reviews prevented project B and C from serious design errors and misapplications of the used technology as occurred in project A. There is no possibility that modern code reviews could not catch these major issues.

By analyzing project B qualitatively and quantitatively, it turned out, that too little human resources were provided until the due day and that lead and cycle times were long, leading to a non-adherence to the deadline. Continuous delivery improved the lead and cycle times, but also more human resources were provided. How large the proportional impact of both changes were, was again quantitatively not determinable. Qualitatively it is clear, that the proportion of continuous delivery on the adherence to the deadline was significant. It was observable that the team developed features and introduced changes more rapidly because the test suite prevented the team from breaking existing functionality most likely. Even if not documented, the test suite prevented the team from delivering failures to the customer very often. The relationship to the customer was therefore also significantly better and the acceptance therefore much easier.

Bibliography

[1] BSA Software Alliance and Economist Intelligence Unit. *The $1 Trillion Economic Impact of Software*. 2017. URL: https://software.org/wp-content/uploads/2017_Software_Economic_Impact_Report.pdf (visited on 04/10/2019) (cit. on p. 1).

[2] BSA Software Alliance and Economist Intelligence Unit. *The Growing EUR 1 Trillion Economic Impact of Software*. 2018. URL: https://software.org/wp-content/uploads/2018_EU_Software_Impact_Report_A4.pdf (visited on 04/10/2019) (cit. on p. 1).

[3] Jorge Aranda, Steve Easterbrook, and Greg Wilson. "Requirements in the wild: How small companies do it." In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. IEEE. 2007, pp. 39–48 (cit. on p. 112).

[4] Roger Atkinson. "Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria." In: *International journal of project management* 17.6 (1999), pp. 337–342 (cit. on pp. 2, 14–17).

[5] Statistik Austria. *Leistungs- und Strukturstatistik ab 2008 - Unternehmensdaten - Hauptergebnisse*. 2016. URL: http://statcube.at/statistik.at/ext/statcube/jsf/tableView/tableView.xhtml (visited on 04/09/2019) (cit. on p. 1).

[6] Alberto Bacchelli and Christian Bird. "Expectations, outcomes, and challenges of modern code review." In: *Proceedings of the 2013 international conference on software engineering*. IEEE Press. 2013, pp. 712–721 (cit. on pp. 61, 112).

[7] Maria Teresa Baldassarre et al. “Harmonization of ISO/IEC 9001:2000 and CMMI-DEV: from a theoretical comparison to a real case application.” In: *Software Quality Journal* 20.2 (June 2012), pp. 309–335. ISSN: 1573-1367. DOI: 10.1007/s11219-011-9154-7. URL: https://doi.org/10.1007/s11219-011-9154-7 (cit. on p. 31).

[8] Sergio Bandinelli et al. “Modeling and improving an industrial software process.” In: *IEEE Transactions on software Engineering* 21.5 (1995), pp. 440–454 (cit. on p. 109).

[9] Paul L Bannerman. “Defining project success: A multilevel framework.” In: *Proceedings of the Project Management Institute Research Conference*. Citeseer. 2008, pp. 1–14 (cit. on p. 16).

[10] Kent Beck and Erich Gamma. *Extreme programming explained: embrace change*. addison-wesley professional, 2000 (cit. on p. 107).

[11] Moritz Beller et al. “Modern code reviews in open-source projects: Which problems do they fix?” In: *Proceedings of the 11th working conference on mining software repositories*. ACM. 2014, pp. 202–211 (cit. on p. 112).

[12] Miklós Biró, János Ivanyos, and Richard Messnarz. “Pioneering process improvement experiment in Hungary.” In: *Software Process: Improvement and Practice* 5.4 (2000), pp. 213–229 (cit. on p. 32).

[13] Christine V Bullen and John F Rockart. “A primer on critical success factors.” In: (1981) (cit. on p. 17).

[14] Victor R Basili-Gianluigi Caldiera and H Dieter Rombach. “Goal question metric paradigm.” In: *Encyclopedia of software engineering* 1 (1994), pp. 528–532 (cit. on p. 31).

[15] Valentine Casey and Ita Richardson. “A practical application of the IDEAL model.” In: *Software Process: Improvement and Practice* 9.3 (2004), pp. 123–132 (cit. on p. 24).

[16] Aileen Cater-Steel and Terry Rout. “SPI long-term benefits: Case studies of five small firms.” In: *Software Process Improvement for Small and Medium Enterprises: Techniques and Case Studies*. IGI Global, 2008, pp. 223–241 (cit. on p. 32).

[17] Fabiano Cattaneo, Alfonso Fuggetta, and Donatella Sciuto. “Pursuing coherence in software process assessment and improvement.” In: *Software Process: Improvement and Practice* 6.1 (2001), pp. 3–22 (cit. on p. 23).

[18] Lianping Chen. “Continuous delivery: Huge benefits, but challenges too.” In: *IEEE Software* 32.2 (2015), pp. 50–54 (cit. on pp. 87, 99).

[19] Lianping Chen. “Continuous delivery: Huge benefits, but challenges too.” In: *IEEE Software* 32.2 (2015), pp. 50–54 (cit. on p. 113).

[20] Tsun Chow and Dac-Buu Cao. “A survey study of critical success factors in agile software projects.” In: *Journal of systems and software* 81.6 (2008), pp. 961–971 (cit. on pp. 17, 19, 87, 105).

[21] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. *CMMI guidlines for process integration and product improvement*. Addison-Wesley Longman Publishing Co., Inc., 2003 (cit. on p. 59).

[22] Bradford K Clark. “Quantifying the effects of process improvement on effort.” In: *IEEE software* 17.6 (2000), pp. 65–70 (cit. on pp. 109, 110).

[23] Paul Clarke and Rory V O’Connor. “The influence of SPI on business success in software SMEs: An empirical study.” In: *Journal of Systems and Software* 85.10 (2012), pp. 2356–2367 (cit. on p. 109).

[24] Daniela Damian and James Chisan. “An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management.” In: *IEEE Transactions on Software Engineering* 32.7 (2006), pp. 433–453 (cit. on p. 111).

[25] Steve Easterbrook et al. “Selecting empirical methods for software engineering research.” In: *Guide to advanced empirical software engineering*. Springer, 2008, pp. 285–311 (cit. on p. 107).

[26] Christof Ebert and Jozef De Man. “Requirements uncertainty: influencing factors and concrete improvements.” In: *Proceedings of the 27th international conference on Software engineering*. ACM. 2005, pp. 553–560 (cit. on p. 59).

[27] Hakan Erdogmus, John Favaro, and Wolfgang Strigel. “ROI in the Software Industry-Return on Investment-Guest Editors’ Introduction.” In: *IEEE Software* 21.3 (2004), pp. 10–11 (cit. on p. 109).

[28] Michael Fagan. “Design and code inspections to reduce errors in program development.” In: *IBM Systems Journal*. Vol. 15. IBM Corp., 1976, pp. 182–211 (cit. on p. 61).

[29] Analia Irigoyen Ferreiro Ferreira et al. “ROI of software process improvement at BL informatica: SPIdex is really worth it.” In: *Software Process: Improvement and Practice* 13.4 (2008), pp. 311–318 (cit. on p. 32).

[30] D Fleck. “A process for very small projects.” In: *Proceedings of the 22nd Annual Pacific Northwest Software Quality Conference*. 2004, pp. 107–115 (cit. on p. 32).

[31] Daniel Galin and Motti Avrahami. “Are CMM program investments beneficial? Analyzing past studies.” In: *IEEE software* 23.6 (2006), pp. 81–87 (cit. on p. 110).

[32] wibas GmbH. *CMMI for development, version 1.3*. Available at `https://www.wibas.com/media/filer_public/2015/05/12/cmmi-dev_v13_poster_v20_kopie.pdf` (2018/11/16). 2013 (cit. on pp. 28, 30).

[33] Christian Printzell Halvorsen and Reidar Conradi. “A taxonomy to compare SPI frameworks.” In: *European Workshop on Software Process Technology*. Springer. 2001, pp. 217–235 (cit. on p. 22).

[34] Donald E Harter, Chris F Kemerer, and Sandra A Slaughter. “Does software process improvement reduce the severity of defects? A longitudinal field study.” In: *IEEE Transactions on Software Engineering* 38.4 (2012), pp. 810–827 (cit. on p. 110).

[35] Donald E Harter, Mayuram S Krishnan, and Sandra A Slaughter. “Effects of process maturity on quality, cycle time, and effort in software product development.” In: *Management Science* 46.4 (2000), pp. 451–466 (cit. on p. 110).

[36] Christian Hofer. “Software development in Austria: results of an empirical study among small and very small enterprises.” In: *Proceedings. 28th Euromicro Conference*. IEEE. 2002, pp. 361–366 (cit. on pp. 2, 105).

[37] Hubert F Hofmann and Franz Lehner. “Requirements engineering as a success factor in software projects.” In: *IEEE software* 4 (2001), pp. 58–66 (cit. on p. 58).

[38] Wei Huang et al. “A novel lifecycle model for Web-based application development in small and medium enterprises.” In: *International Journal of automation and computing* 7.3 (2010), pp. 389–398 (cit. on p. 32).

[39] Jez Humble and David Farley. *Continuous delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley Boston, 2011 (cit. on pp. 87, 88).

[40] CMMI Institute. *Do ISO Standards And CMMI Work Together?* Available at `https://cmmiinstitute.zendesk.com/hc/en-us/articles/115004587567-Do-ISO-standards-and-CMMI-work-together-` (2018/11/16) (cit. on p. 31).

[41] Richardson Ita and Chrisiane Gresse von Wangenheim. "Why Are Small Software Organizations Different?" In: *IEEE software 24* 1 (2007), pp. 18–22 (cit. on pp. 1, 2, 8, 32, 105).

[42] Jan Wiedemann Jacobsen et al. "On the role of software quality management in software process improvement." In: *International Conference on Product-Focused Software Process Improvement*. Springer. 2016, pp. 327–343 (cit. on p. 20).

[43] Capers Jones. *Programming Productivity*. New York, NY, USA: McGraw-Hill, Inc., 1986. ISBN: 0-07-032811-0 (cit. on p. 49).

[44] Stephen H Kan. *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002 (cit. on pp. 21, 31, 44, 45, 48–51).

[45] Mark Keil et al. "A framework for identifying software project risks." In: *Communications of the ACM* 41.11 (1998), pp. 76–83 (cit. on pp. 59, 60).

[46] Martin Kropp et al. "Satisfaction, practices, and influences in agile software development." In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. ACM. 2018, pp. 112–121 (cit. on p. 113).

[47] Marco Kuhrmann, Philipp Diebold, and Jürgen Münch. "Software process improvement: a systematic mapping study on the state of the art." In: *PeerJ Computer Science* 2 (2016), e62 (cit. on pp. 2, 20, 32).

[48] Felicia Kurniawati and Ross Jeffery. "The long-term effects of an EPG/ER in a small software organisation." In: *2004 Australian Software Engineering Conference. Proceedings.* IEEE. 2004, pp. 128–136 (cit. on pp. 2, 109).

[49] Claude Y Laporte, Simon Alexandre, and Rory V O'Connor. "A software engineering lifecycle standard for very small enterprises." In: *European Conference on Software Process Improvement*. Springer. 2008, pp. 129–141 (cit. on p. 1).

[50] Brian Lawrence, Karl Wiegers, and Christof Ebert. "The top risk of requirements engineering." In: *IEEE Software* 18.6 (2001), pp. 62–63 (cit. on p. 58).

[51] Annabella Loconsole. "Empirical studies on requirement management measures." In: *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. IEEE. 2004, pp. 42–44 (cit. on p. 59).

[52] Mika V Mäntylä and Casper Lassenius. "What types of defects are really discovered in code reviews?" In: *IEEE Transactions on Software Engineering* 35.3 (2009), pp. 430–448 (cit. on p. 112).

[53] Fergal Mc Caffery, Philip S Taylor, and Gerry Coleman. "Adept: A unified assessment method for small software companies." In: *IEEE software* 24.1 (2007) (cit. on p. 33).

[54] Thomas J McCabe. "A complexity measure." In: *IEEE Transactions on software Engineering* 4 (1976), pp. 308–320 (cit. on p. 50).

[55] Fergal McCaffery, Donald McFall, and F George Wilkie. "Improving the express process appraisal method." In: *International Conference on Product Focused Software Process Improvement*. Springer. 2005, pp. 286–298 (cit. on p. 33).

[56] Bob McFeeley. *IDEAL: A User's Guide for Software Process Improvement.* Tech. rep. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1996 (cit. on p. 24).

[57] Shane McIntosh et al. "An empirical study of the impact of modern code review practices on software quality." In: *Empirical Software Engineering* 21.5 (2016), pp. 2146–2189 (cit. on p. 113).

[58] Shane McIntosh et al. "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects." In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM. 2014, pp. 192–201 (cit. on p. 113).

[59] Nils Brede Moe et al. “Process guides as software process improvement in a small company.” In: *Proceedings of the EuroSPI Conference, Germany*. 2002 (cit. on pp. 2, 109).

[60] Mariano Montoni and Ana Regina Rocha. “A methodology for identifying critical success factors that influence software process improvement initiatives: an application in the Brazilian software industry.” In: *European Conference on Software Process Improvement*. Springer. 2007, pp. 175–186 (cit. on p. 32).

[61] Mahmood Niazi and Muhammad Ali Babar. “Identifying high perceived value practices of CMMI level 2: an empirical study.” In: *Information and software technology* 51.8 (2009), pp. 1231–1243 (cit. on p. 33).

[62] Mahmood Niazi, Muhammad Ali Babar, and Suhaimi Ibrahim. “An empirical study identifying high perceived value practices of CMMI level 2.” In: *International Conference on Product Focused Software Process Improvement*. Springer. 2008, pp. 427–441 (cit. on p. 33).

[63] Bashar Nuseibeh and Steve Easterbrook. “Requirements engineering: a roadmap.” In: *Proceedings of the Conference on the Future of Software Engineering*. ACM. 2000, pp. 35–46 (cit. on p. 59).

[64] Rory V O’Connor and Gerry Coleman. “Ignoring’Best Practice’: why Irish software SMEs are rejecting CMMI and ISO 9000.” In: (2009) (cit. on p. 32).

[65] Richard Paul Olsen. “Can project management be defined?” In: Project Management Institute. 1971 (cit. on p. 14).

[66] César Pardo et al. “Homogenization of Models to Support multi-model processes in Improvement Environments.” In: *4th International Conference on Software and Data Technologies ICSOFT*. Vol. 9. 2009, pp. 151–156 (cit. on p. 31).

[67] Francisco J Pino et al. “Harmonizing maturity levels from CMMI-DEV and ISO/IEC 15504.” In: *Journal of Software Maintenance and Evolution: Research and Practice* 22.4 (2010), pp. 279–296 (cit. on p. 31).

[68] Alcides Quispe et al. “Requirements engineering practices in very small software enterprises: A diagnostic study.” In: *Chilean Computer Science Society (SCCC), 2010 XXIX International Conference of the*. IEEE. 2010, pp. 81–87 (cit. on pp. 8, 39, 40, 58, 112).

[69] Jane Radatz, Anne Geraci, and Freny Katki. “IEEE standard glossary of software engineering terminology.” In: *IEEE Std* 610121990.121990 (1990), p. 3 (cit. on p. 45).

[70] Olli Rissanen and Jürgen Münch. “Transitioning towards continuous delivery in the B2B domain: a case study.” In: *International Conference on Agile Software Development*. Springer. 2015, pp. 154–165 (cit. on p. 113).

[71] Janne Ropponen and Kalle Lyytinen. “Components of software development risk: How to address them? A project manager survey.” In: *IEEE transactions on software engineering* 26.2 (2000), pp. 98–112 (cit. on p. 58).

[72] Mark von Rosing, Henrik von Scheel, and August-Wilhelm Scheer. *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM, Volume I*. Morgan Kaufmann Publishers Inc., 2014 (cit. on p. 2).

[73] Terence P Rout and Angela Tuffley. “Harmonizing iso/iec 15504 and cmmi.” In: *Software Process: Improvement and Practice* 12.4 (2007), pp. 361–371 (cit. on p. 31).

[74] Marty Sanders. *The SPIRE Handbook: Better Faster Cheaper Software Development in Small Organisations*. Centre for Software Engineering, Limited, 1998 (cit. on p. 32).

[75] Marty Sanders and Ita Richardson. “Research into long-term improvements in small-to medium-sized organisations using SPICE as a framework for standards.” In: *Software Process: Improvement and Practice* 12.4 (2007), pp. 351–359 (cit. on p. 32).

[76] Christian Schindler. “Agile software development methods and practices in Austrian IT-industry: Results of an empirical study.” In: *2008 International Conference on Computational Intelligence for Modelling Control & Automation*. IEEE. 2008, pp. 321–326 (cit. on p. 2).

[77] Anna Schmitt and Philipp Diebold. “Why do we do software process improvement?” In: *International Conference on Product-Focused Software Process Improvement*. Springer. 2016, pp. 360–367 (cit. on pp. 18, 20).

[78] L Scott, R Jeffery, and U Becker-Kornstaedt. “Preliminary results of an industrial EPG evaluation.” In: *Proc. 4th ICSE Workshop on Software Engineering over the Internet, Toronto Canada*. 2001, pp. 12–19 (cit. on p. 109).

[79] Louise Scott et al. “Understanding the use of an electronic process guide.” In: *Information and Software Technology* 44.10 (2002), pp. 601–616 (cit. on p. 109).

[80] Erik Simmons. “Lessons Learned in Five Years of Requirements Engineering Improvement.” In: *presentation at RE Day* 13 (2005) (cit. on p. 111).

[81] Harry M Sneed. *Software-Management*. Müller Köln, 1987 (cit. on p. 16).

[82] DM Rini van Solingen and Egon W Berghout. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. McGraw-Hill, 1999 (cit. on pp. 25, 31, 32, 34).

[83] Daniel Ståhl and Jan Bosch. “Modeling continuous integration practice differences in industry software development.” In: *Journal of Systems and Software* 87 (2014), pp. 48–59 (cit. on p. 88).

[84] International Organization for Standardization. *ISO 9001:2015*. Available at `https://www.iso.org/files/live/sites/isoorg/files/standards/docs/en/iso_9001.pptx` (2018/11/15). 2015 (cit. on p. 25).

[85] International Organization for Standardization. *Quality management principles*. Available at `https://www.iso.org/files/live/sites/isoorg/files/archive/pdf/en/pub100080.pdf` (2018/11/15). 2015 (cit. on p. 26).

[86] David J Storey. *Entrepreneurship and new firm*. Routledge, 1982 (cit. on p. 32).

[87] Martyn Thomas and Frank McGarry. “Top-down vs. bottom-up process improvement.” In: *IEEE Software* 11.4 (1994), pp. 12–13 (cit. on p. 22).

[88] Axel Van Lamsweerde. “Requirements engineering: from craft to discipline.” In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM. 2008, pp. 238–249 (cit. on p. 58).

[89] Rini Van Solingen. “Measuring the ROI of software process improvement.” In: *IEEE software* 21.3 (2004), pp. 32–38 (cit. on p. 109).

[90] Linda Wallace and Mark Keil. “Software project risks and their effect on outcomes.” In: *Communications of the ACM* 47.4 (2004), pp. 68–73 (cit. on pp. 40, 59).

[91] Christiane Gresse von Wangenheim et al. “Experiences on establishing software processes in small companies.” In: *Information and software technology* 48.9 (2006), pp. 890–900 (cit. on pp. 2, 108, 111).

[92] Karl Wiegers. “Process Improvement that Works.” In: *Software Development* 7.10 (1999), pp. 24–30 (cit. on p. 21).

[93] F George Wilkie, Donald McFall, and Fergal McCaffery. “An evaluation of CMMI process areas for small-to medium-sized software development organisations.” In: *Software Process: Improvement and Practice* 10.2 (2005), pp. 189–201 (cit. on p. 33).

[94] Chanwoo Yoo et al. “A unified model for the implementation of both ISO 9001: 2000 and CMMI by ISO-certified organizations.” In: *Journal of Systems and Software* 79.7 (2006), pp. 954–961 (cit. on p. 31).