Adopting agile software development in a multi-project environment.

Master Thesis
at
Graz University of Technology

submitted by
0430819 Elisabeth Richter, Ing. Bakk. rer. soc. oec.

in
924 Software Development and Business

Institut for General Management and Organization
(UFO),
Graz University of Technology
A-8010 Graz

January 12, 2010

This paper is written in english.

Supervisor:	Wiss.-Ass. Dipl.-Ing. Ernst Stelzmann
External Advisor:	Dipl.-Ing. Gregor Karlinger (XiTrust Secure Technologies GmbH)
Assessor:	o.Univ.-Prof. Dipl.-Ing. Dr.sc.techn. Reinhard Haberfellner

STATUTORY DECLARATION

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 marked all material which has been quoted either literally or by content from the used sources.

....

date (signature)

Acknowledgements

Now, after finishing this thesis, I want to gratefully thank everyone who supported me and accompanied me along the way. Their support and encouragement was invaluable especially during some rougher times.
First, I would like to thank all the people at XiTrust and the UFO institute. They openly welcomed me and put a lot of trust in my abilities. Special thanks go to Georg Lindsberger who commissioned me not only to write this thesis but also two other papers for his organization. He and Gregor Karlinger dedicated many precious hours to productive discussions about the ongoing work for this thesis. I would also like to thank Ernst Stelzmann for his input and for trusting me in organizing my work without tight control. Further thanks go to Florian Simon for his input from practice.
Additionally, completing my studies would have been a lot more difficult without the support of many other people in my life. First of all, a huge thank you goes to my husband Wolfgang. Thanks for your love, support and encouragement. Thanks for coping with me being deeply immerged in my studies at times. Thanks also go to my parents for supporting me mentally and financially. I would also like to thank my grandparents and siblings for being so proud of me. To complete the family, thanks go to my unborn baby for not upsetting my stomaching during writing this thesis.
Finally, my fellow students and friends deserve to be mentioned here. Thank you for all the work and courses we have completed together and apologies for my sometimes commanding tone. Further thanks go to all people who contributed to my degree, but are not mentioned by name.

Elisabeth Richter
Graz, Austria, Dezember 2009

ABSTRACT

Software development is a complex endeavor. The increasingly demanding and fast moving market adds even more complexity. In this setting software development organizations often struggle to develop products satisfying customer demands. Agile Methodologies promise to resolve the situation, if used orderly.

This thesis describes the adoption process of agile software development at XiTrust Secure Technologies GmbH. After determining the company-specific requirements appropriate methodologies were selected. During the adoption process special attention was paid to comprehension and sustainability of the new techniques.

A special challenge was the multi-project environment. At XiTrust Secure Technologies GmbH, usually a couple of small projects are executed concurrently. Results are, very small team sizes and an unevenly distributed workload. Hence, coordination and synergies between projects had to be expanded to successfully adopt agile software development in this context. All process decisions were recorded in an organization manual for XiTrust.

KURZFASSUNG

Das Entwickeln von Software ist eine komplexe Aufgabenstellung. Zusätzliche Komplexität entsteht durch den immer anspruchsvolleren und schnelllebigeren Softwaremarkt. Vor diesem Hintergrund haben Unternehmen oft Probleme ihre Softwareprodukte zur Zufriedenheit der Kunden zu entwickeln. Agile Methoden versprechen diese Probleme zu lösen, wenn sie richtig angewendet werden.

Diese Diplomarbeit beschreibt wie agile Softwareentwicklung bei XiTrust Secure Technologies GmbH eingeführt wurde. Grundsätzlich galt es, die firmenspezifischen Anforderungen zu bestimmen und bei der Methodenwahl und Prozesskonzeption zu berücksichtigen. Bei der Einführung der neuen Methoden wurde besonders auf Nachhaltigkeit und Verständnis der neuen Techniken geachtet.

Eine besondere Herausforderung war die Multiprojektumgebung. XiTrust Secure Technologies GmbH wickeln üblicherweise mehrere Kleinprojekte parallel ab. Neue Koordinationsmöglichkeiten und Synergien wurden geschaffen, um auch in diesem Kontext einen Erfolg mit agiler Softwareentwicklung zu ermöglichen. Alle Prozessentscheidungen wurden in einem Organisationshandbuch für XiTrust dokumentiert.

Contents

Chapter 1

Introduction

The software market is often described as increasingly demanding and fast moving. Software is outdated as fast as the hardware it operates on improves. Changing markets or new government regulations force organizations and thereby their software vendors to react quickly with new software versions or enhancements. In this setting many software development companies struggle to finish their software development endeavors with satisfactory results for themselves and their customers.

The same was true for XiTrust Secure Technologies, the organization this thesis was developed for. Time and again XiTrust projects finished late or over budget. Problems in one project cascaded into problems in subsequent projects. Additionally, most projects were staffed with one or two employees only, resulting in an unevenly distributed workload and troubles with resource scheduling.

As a countermeasure XiTrust decided to introduce a more orderly approach to its software development. Scrum was deemed suitable and an adoption attempt started. Unfortunately, the adoption process was unstructured and a responsible person was not assigned. Consequently, results where meager and discouraging. Then, the author was commissioned to overtake the adoption of Scrum and to select an appropriate set of methodologies within the scope of this thesis.

1.1 Organization

The organization this thesis was written for is an SMB located in Graz, Austria. It was founded in 2002 and named XiCrypt Internetsicherheitslösungen GmbH (translated: XiCrypt Internet Security Solutions GmbH). In 2009 it was renamed to XiTrust Secure Technologies GmbH and now has a staff of ten. It's legal form is a limited liability partnership, in German GmbH (Gesellschaft mit beschränkter Haftung). The internal organization is rather flat with loose functional grouping into development and soft-

ware support, sales and customer relations, and supporting functions.

XiTrust's core competence is to create and verify digital signatures. Based thereon a product line is developed which optimizes and secures their customer's communication channels and processes. The products by name are the XiTrust Business Server, also known as e3server, the XiTrust Interaction Server and the XiTrust Timestamp Server. Additionally special solutions for customers are developed on request.

Most XiTrust customers are midsized and big businesses, not restricted to a specific industry. XiTrust solutions are for example already applied in health insurance, construction industry and at utility companies. Amongst their customers are also many public authorities.

XiTrust maintains a close partnership with academic institutions, such as the Graz University of Technology and Campus02 University of Applied Sciences, to advance its research and development efforts. Regularly research projects are conducted with students from those institutions. XiTrust staff also teaches at several academic institutions in Graz. Additionally, an internationally active partner network to provide integral solutions to customers has been build up.

1.2 Objectives

Based on the issues and challenges already known four main goals were agreed on:

- Increased customer satisfaction through improved reliability and adherence to delivery dates.
- Earlier project completion through improved project progress and control to optimize cost and payment situation.
- Broadened documentation of decisions and lessons.
- Increased empowerment of each employee.

To achieve listed goals two main fields of operation were identified. First, the organization and projects are to be analyzed to determine the situation at hand. Then, the specific areas of improvement and demands on methodologies are to be derived.

Second, a methodology, preferably agile, and adoption process for XiTrust are to be selected. This includes studying relevant literature about agile methodologies and their adoption in organizations. Additionally, it was already decided to experiment with Scrum. The author was appointed the role of the scrum master (see section 3.1). If necessary, agile methodologies are to be combined with traditional approaches to serve the special needs at XiTrust.

1.3 Outline

This thesis describes the adoption process of agile methodologies at XiTrust. In chapter 2 an overview of agile software development is given. Its origins and history including the Manifesto of Agile Software Development are presented. Common elements and concepts as well as the context to be considered when adopting agile software development provide the necessary background for specific methodologies.

Chapter 3 details the methodologies relevant for this thesis. Five methodologies were selected, three of them agile, namely Scrum, XP, and Lean Software Development. The other two, Goal Directed Project Management and change project management are not classified as agile although Goal Directed Project Management shows some similarities.

The development procedure of the processes as well as the rationale behind the selection of the methodologies are detailed in chapter 4. The situation at XiTrust and at the outset of the project experiment is depicted. Important events and factors contributing to the end result are specified.

Chapter 5 summarizes the most important findings. Additionally, important factors for how to make the most of the now established processes and areas for further improvements are identified. Appendix A holds the organization manual for XiTrust, which includes all process description developed in this thesis. It is written in German to enable ongoing enhancements within the company.

Chapter 2

Agile Software Development

Agile Software Development is at the heart of a controversial debate since it emerged. Some swear by it and appear almost dogmatic in their attempts to win new followers. Others deem it chaotic and a hack-and-fix approach with a nice name to conceal its inadequacies. Those are certainly extreme and antipodal positions which leave a lot of middle ground.

When pursuing the idea to implement Agile Software Development in an organization, it is advisable to carefully analyze both the methodologies and the own home ground. Agile Software Development has its weaknesses just as, what is known as, traditional software development. A careful analysis enables utilizing the best of both worlds, agile and traditional, for the situation at hand (Boehm, 2002).

2.1 Origins

The origin of agile software development is usually dated around the year 2000 (Boehm, 2006). While this is certainly true for the naming, for the incorporated concepts it is not. Iterative and incremental software development approaches are in use since around 1960. Since then a series of large software development projects were completed using iterative and incremental approaches including life critical systems, for instance a submarine control system. Many of those projects used practices now known to be part of agile software development, such as test-first, short feedback loops, customer involvement and time-boxed iterations. (Larman and Basili, 2003, P. 1)

But agile software development also draws from other well-studied disciplines. The idea of self-organizing around a few rules is based on the theory of complex adaptive systems (Highsmith, 2002, P. XXV). Using feedback for process control goes back to the science of empirical process control (Schwaber, 2004, P. 2). Some methodologies are based on proven methodologies from other industries. For example, Lean Devel-

opment and Lean Software Development use ideas from Lean Production and Lean Product Development which in turn uses concepts from queuing theory (Poppendieck and Poppendieck, 2007).

2.1.1 Emerging Methodologies

Based on the success of those projects a series of methodologies evolved over time. Interestingly enough, the waterfall model is now deemed a misinterpretation of one first descriptions of iterative and incremental development by Winston Royce in 1970 (Larman and Basili, 2003, P. 2). In the following decades companies mostly used their very own construct of iterative and incremental development. In publications names such as Adaptive Development and Evolutionary Prototyping where used. A landmark publication at that time was Barry Boehm's Spiral Model, a very formalized, risk driven iteration model.

In the 1990s the effort to formalize iterative and incremental development strengthened. Groups and individuals summarized their project experiences in formal methodology descriptions. For example, during the C3 (Chrysler Comprehensive Compensation System) project eXtreme Programming (XP) formed and Feature Driven Development (FDD) has its origins in development of a large logistics software system in Singapore. The Dynamic Systems Development Method (DSDM) was the result of the effort of a group of rapid application developers. Alistair Cockburn studied a vast amount of successful software projects and packed his findings into the Crystal Methodologies. Similar stories are the basis of Scrum and other methodologies.

2.1.2 The Agile Manifesto

The efforts to formalize and promote those methodologies reached a peak with formation of the Agile Alliance, a group of seventeen supporters including the founders of XP, Scrum and more. At the beginning of 2001, they convened and agreed on a common ground. They coined the name Agile Software Development and recorded their agreement as the Manifesto of Agile Software Development.

Agile software development certainly stems from iterative and incremental software development, but it adds a strong support for certain basic values such as Respect, Courage, Openness, Focus, Commitment, Feedback, Communication and similar. In the Agile Manifesto (Agile Alliance, 2001) those values are reflected as

- *'Individuals and interactions over processes and tools'*
- *'Working software over comprehensive documentation'*

- *'Customer collaboration over contract negotiation'*
- *'Responding to change over following a plan'*

Those four statements express, in an agile software development environment there is some value in the items on the right, but the items on the left are valued more. In addition to those four values the manifesto states twelve principles to elaborate the values.

2.2 Common Elements

Since agile software development methodologies have a common history, it is hardly surprising, they share common elements. The next sections detail those elements to provide a backdrop for the description of specific methodologies in chapter 3.

2.2.1 Planning Levels

As the Agile Manifesto states, planning is part of agile software development. The difference is, in order to stay responsive planning occurs on different levels for different time horizons. Detailed plans are created for the near future only while a coarse structure guides the overall development direction. That strategy is often depicted as the planning onion, as shown in figure 2.1.

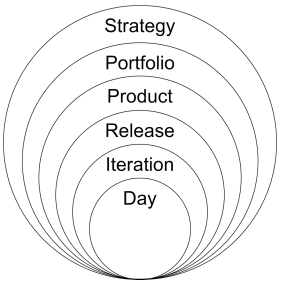

Figure 2.1: The Planning Onion (Cohn, 2006, P. 28)

Agile teams plan for the three innermost levels of the planning onion. **Release planning** determines schedule, scope and resources for a project. Scope definition often includes the definition of a minimum feature set which is the set of features which absolutely has to be included in a release to make it valuable for a customer. Release plans are created at the start of a project and updated at the end of each iteration.

Iteration planning is more detailed than release planning. Based on the last iteration and the release plan, work for the iteration about to be started is selected. Individual tasks for transforming a requirement into deployable software are defined.

Daily planning synchronizes the efforts of the individual team members and resolves dependencies on a daily basis. Individual activities are coordinated and impediments uncovered and removed. The planning horizon is no longer than a single work day.

The three other levels, product, portfolio and strategy, are planned in the broader context of the whole organization. **Product planning** lays out the cornerstones of the developed software product beyond the current release. **Portfolio planning** coordinates the organization's selection of developed products and **strategic planning** provides a vision for the focus of the whole organization. (Cohn, 2006, P. 28f)

2.2.2 Roles

Collaboration and appropriate contact with the people involved in software development is at the heart of each agile methodology. Hence, a couple of roles are required for a truly agile methodology.

Customer. The customer sets budget boundaries and states vision, purpose and goals for a project which is also known as product mission. He ultimately decides what is of value for him and has to be included in the software product. The customer is usually external to the development organization or department and commissions a software development project.

The customer or a designated representative is obliged to actively participate during the whole project lifetime. Without a customer willing to provide resources for participation only mediocre results are to be expected due to the lack of customer feedback.

Enduser. Usually a software product is not used by the customer. Often the consequence is, the needs of the people actually using the software, the enduser, are not completely and correctly understood. Therefore, it is vital to include enduser in the feedback loops.

Team. All project participants work together as team. Hence, an agile team is interdisciplinary. Team size is usually 5 - 8 people. Team members organize and control their work based on a common goal and plans made. They are empowered to make their own decisions about how to provide the best solution for the customer.

Leader. Despite the rather flat hierarchy of Agile Software Development usually one person provides leadership for certain aspects of the development. The person may or may not be assigned as leader, but guides the team with superior knowledge about market needs, process, customer, technology or others. The existence of leadership focuses the development efforts in a way not achievable through

management. Often this role is referred to as *champion* to express it must not be an appointed position.

2.2.3 Concepts

Agile software development is complex therefore it is hard to reduce it to a few common concepts. Nevertheless, to gain a deeper understanding, five core concepts are expounded.

Iterative and Incremental. In simple words, iterative and incremental means delivering a deployable product after a predetermined period of time. The concept of being iterative is rather easy. Basically, it can be compared to a project milestone reoccurring after a fixed, always identical period of time.

Being incremental is not complex, but often hard to implement. After each iteration, a deployable software product based on the last increment has to be ready. This requires focus on a common goal, appropriate planning methods, technical excellence and practices such as continuous integration in place.

Adaptive. Adaptability is to be responsive to change. Flexibility, unachievable through sticking to predefined plans, is required. Constant and early feedback allows for revising the plans and reacting according to the current situation and need.

Additionally, it means to create change. Some situations require striking new paths not immediately evident in feedback. For example, ceasing product acceptance might indicate the need for a new product, not a more aggressive marketing strategy. This aspect is often neglected in criticism, agile is nothing more than reacting to bad situations.

(Highsmith, 2002, P. 29) sums it up in his favorite description of agility:

*'Agility is the ability to both **create** and **respond** to change in order to profit in a turbulent business environment.'*

People Centered. *'People trump process'* (Highsmith and Cockburn, 2001b) expresses excellent people will accomplish their assignment no matter which process is in place, but no process will make up for knowledge and skill inadequacies. The agile approach is to use a set of generative rules as guidance and then rely on people to find creative solutions for complex problems as they arise (Highsmith and Cockburn, 2001a).

In addition, an important aspect of being people centered is the focus on tacit knowledge (Highsmith, 2002, P. XXVII). Tacit knowledge is internal or, in other words, inside people's heads. It is build up through training and own or learned previous experiences. It is automatically combined and applied when coming

across similar situations as opposed to explicit knowledge provided by extensive process descriptions trying to provide for every situation.

Feature Centered. One of the main principles is to deliver as much value as possible to the customer. Hence, planning is feature centered. All plans use features as their main work unit since features, not technical tasks, pose value to the customer. (Cohn, 2006, P. 12)

Moreover, the features are defined customer-oriented. They are written together with the customer or at least from the customer's perspective without using technical jargon. Sometimes stories, how a user will interact with the system in question, are written. Thus, a feature specification is often referred to as a user story. Specifications that describe a bigger part of a system are called epics. Themes are used to group stories and epics by topic or other categories.

Visual Workplace. People best thrive in a pleasant workplace. A mixture of public and private space allows for communication and collaboration as well as undisturbed work. Besides, important information should be ready at one glance to reduce decision delays.

Agile software development suggests using, what is known as, information radiators and signals. Information radiators are boards and displays in the workplace used to provide important data about the current assignment. Data usually shown are the plan for the current iteration, software metrics, team metrics, design and architecture diagrams and more. Signals are used to call immediate attention to a problem. Many teams use fun signals such as lava lamps or traffic lights to signal a broken build.

2.3 Context

Just as any other methodology, agile methodologies need to be applied in the right context. Some contexts are more suitable or ready than others. Important properties in the context of an adoption process are:

Organization. A software development project is embedded in an organization. Most organizations, especially larger, use a certain amount of rules and regulations. A high number of strictly enforced rules and regulations hinder adoption of an agile methodology. Besides, some rules contradict certain practices. For example, delaying the approval of large projects until the annual budgeting is in conflict with establishing an even, steady flow of work (Poppendieck and Poppendieck, 2007, P. 103).

Knowledge. (Highsmith and Cockburn, 2001a) states people and their knowledge, skills and abilities are the primary drivers of project success. In an agile environ-

ment many degrees of freedom are available to those people. Hence, personnel need to know how to react to new situations without the guidance of rules and regulations. A high percentage of highly educated professionals are required. (Boehm and Turner, 2003)

Type of Projects. Project aspects such as size, life criticality, technology, stability and elaborateness of requirements affect the applicability of and the need for agile methodologies. For example, a project developing a new version of a legacy system usually has rather exact requirements derived from the old software and might be developed within a more regulated process setting.

Focus. Different agile methodologies focus on different aspects of software development. Therefore, a certain maturity of the other aspects is required. In Kent Beck's words, constant change is not possible unless continuous integration, testing, coding standards and more are in place (Beck, 2000, P. 66).

Every organization is a unique context for the application of agile software development. Often, one or more aspects inside this context are not originally ready for the application of a specific agile methodology. Therefore, it is not unusual to combine two or more agile methodologies to form a holistic approach for the organizations specific software development environment. Many experience reports give an account of this practice. For example, (Kniberg, 2007) describes their combination of Scrum and XP.

Chapter 3

Selective Methodologies

For both, professional project management and software engineering a vast repertory of different methodologies is available to choose from. The challenge is to find the ones applicable to a specific project and software development endeavor. A proven methodology will fail if used in the wrong context even if perfectly applied.

The methodologies presented in this chapter have been analyzed and found applicable for the situation at hand at XiTrust. Selection was based on a number of criteria:

- Available knowledge
- Challenges at hand
- Customer project execution requirements

A more detailed rationale for methodology selection is given in chapter 4.

3.1 Scrum

Scrum is, besides eXtreme Programming, by far the best known agile methodology. A quick internet search reveals, a high number of companies use Scrum. Amongst them are big names such as Google, Microsoft, Sun, IBM and more who successfully introduced Scrum. Nevertheless, beside this impressive record of successful Scrum implementations, Scrum has also a reputation of failing frequently (Shore, 2008). Reasons for failure, as well as an overview and benefits of Scrum are detailed in the following paragraphs.

3.1.1 Overview

Scrum applies the theory of empirical process control to software development. The three cornerstones of empirical process control are: visibility, inspection and adaption. The aspects of the process affecting the end result must be visible to those controlling the process. Deceiving appearances will lead to wrong adaptations. Inspections occur as frequent as necessary to catch undesired variances early. All artifacts are inspected with the appropriate participants. If it is learned through inspection one or more aspects of the process are deviating and jeopardizing the desired end result, adaptations are made. (Schwaber, 2004, P. 2ff)

To implement empirical process control, Scrum comprises only a few concise rules. A scrum project is executed using a series of short timeboxed cycles. These cycles are called sprints in Scrum or more general, iterations. Each project requires a single product backlog, the scrum master, the product owner and the team to be ready for work.

A sprint starts with the sprint planning meeting. A small work package is splitted off the product backlog forming the sprint backlog. During the sprint the scrum team transforms the requirements in the sprint backlog into working, deployable software, the product increment. Each day the team meets to discuss planned tasks and impediments in, what is known as, the daily scrum.

Each sprint is concluded by two meetings. In the sprint review the team presents the product owner and customer the product increment. They approve if the increment is in compliance with the customer's ideas. Secondly a moderated workshop, the sprint retrospective, is conducted to reflect about process, teamwork and other relevant topics. The results are improvement experiments fed into the next sprint, thereby closing the cycle. (Pichler, 2008, P. 7f)

Listed elements and cycles are depicted in figure 3.1. Named roles, artifacts, meetings and processes are detailed below.

3.1.2 Roles

As already stated above, in Scrum three roles are known: product owner, scrum master and team. These three roles tightly collaborate to successfully complete their software development endeavor.

Product Owner. The product owner is an in-house customer representative. He defines and manages the requirements. The requirements are then prioritized by customer business value. Using requirements and priorities, the product owner

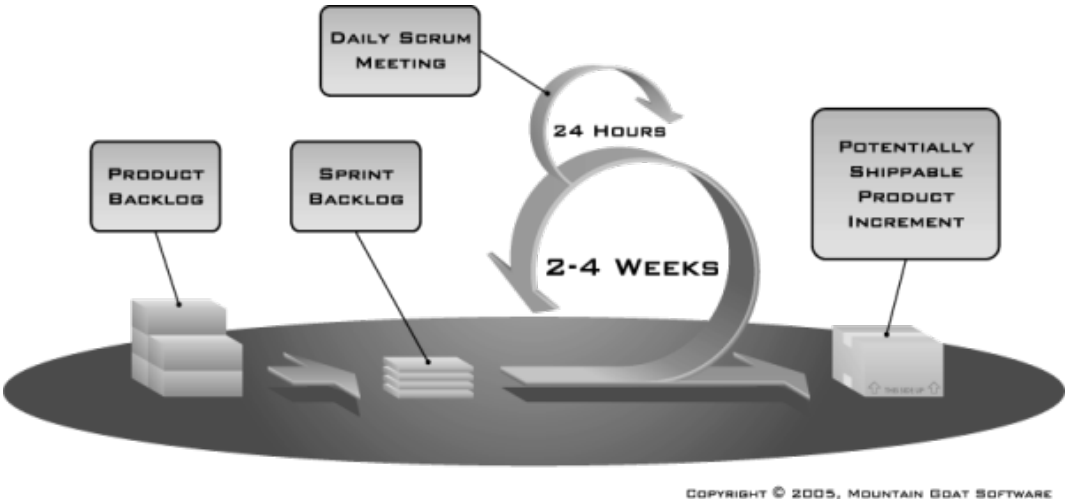

Figure 3.1: Scrum Overview (Mountain Goat Software, 2005)

builds and maintains a release plan. Performing listed tasks, he is responsible for maximizing the customer's return of investment.

Often the product owner is referred to as the 'single wringable neck' because his actions decide between success and failure. All his work should be done in close collaboration with the customer and the team. Therefore, the product owner must be empowered to make decisions regarding the requirements and project course in order not to delay important decisions. Additionally, he must be available to work with the team and customer each single day to clarify requirements and check work results. (Pichler, 2008, P. 9ff)

Team. A scrum team is interdisciplinary. Five to nine programmers, software architects, software designer, database developer, tester and individuals of other professions collaborate to create working software and other software related artifacts. It is beneficial if team members are assigned to exactly one project at once otherwise overhead of the different meetings might be too high and team cohesion will suffer.

A scrum team is empowered, self-organized, autonomous and collocated. It is empowered to decide how many requirements are put into the spring backlog. Organizing and deciding how to implement the sprint backlog is also to the discretion of the team. Furthermore, it should be as autonomous as possible from external dependencies, such as experts outside the team, in achieving the sprint goal. Finally, all team members are collocated in a designated team area or team room to facilitate spontaneous and broadband communication. (Pichler, 2008, P. 13ff)

Scrum Master. The scrum master's main duties are to support and coach the team and the product owner. At the outset of a scrum implementation he establishes the new rules and coaches all parties to use Scrum to the best of their abilities. He facilitates communication inside the team and with the product owner. He pro-

tects the team from spontaneous requests beyond the sprint backlog and removes impediments in their day to day work.

The scrum master role might be difficult to manage at first since more often than not it is very much different from what one knows to date. A scrum master supports the team, but may not give instructions. In other words, he has no authority but may influence the team by serving their needs. For a scrum master it is beneficial to know facilitation techniques, to have good communication, conflict management and collaborative decision making skills and experience in software development and application of agile development practices. (Pichler, 2008, P. 19ff)

In conclusion, the differentiation between product owner and team corresponds to the What vs. How Dilemma (Davis, 1993, P. 17f). The product owner defines the problem and the team finds a solution. The scrum master facilitates the process and fosters smooth progress.

3.1.3 Concepts

Scrum introduces three special concepts to organize, plan and control development work. Once learned and understood they are rather transparent and effective.

Backlogs

The backlogs are queues of open work items. A minimal specification of a work item comprises a unique identifier, a description, an estimate and a priority. Different work items are used in different backlogs.

Product Backlog. The product backlog holds all required work for software development. Product backlog items are described from a customer's perspective. All items must be intelligible by and add value for the customer. Work items typically found in a product backlog are themes, epics and user stories.

Sprint Backlog. The sprint backlog contains the amount of work planned for a single sprint. Sprint backlogs are not filled in advance, but at the very start of each sprint. Items in a sprint backlog are user stories and tasks. Tasks detail a story from the technical perspective and are usually estimated in hours. Task priorities may be derived from dependencies between tasks or process requirements. For example test tasks have to be completed first if Test Driven Development is used.

Impediment Backlog. Impediments block the team's progress or faster development. It is the scrum masters responsibility to resolve them as quickly as possible.

Hence, the number of impediments, the work items in the impediment backlog, should stay low.

Backlogs should be kept short and not filled thoughtlessly. Preliminary work on items with low priority is to be limited since the probability is high they are changed or thrown away.

Burndown Charts

The burndown charts are used for progress monitoring. At predefined points they are updated, actual progress checked against anticipated progress and acted upon. A burndown chart usually exhibits five properties: time frame on the X-axis, amount of work to be done on the Y-axis, ideal and actual burndown and a trend line for the actual burndown. At the point the trend line intersects the X-axis the workload amounts to zero. This line helps to anticipate at what point all work is completed.

Most Scrum implementations use two different burndown charts. Figure 3.2 and 3.3 show example burndown charts as further explained below.

The **Sprint Burndown Chart** is used to measure progress during a sprint. The X-axis corresponds to the timebox of a sprint, the Y-axis to the number of hours estimated for tasks. Updates occur daily or more frequently. Progress is monitored at the daily scrum. If it becomes evident the commitment is not achievable requirements of lower priority are nominated for removal. If progress is faster than anticipated new requirements may be added. Both has to be done in collaboration with the product owner. Figure 3.2 shows a sprint burndown chart including a line for the team's capacity (Pichler, 2008, P. 117f).

The **Release or Product Burndown Chart** shows progress towards finishing the software product or a specific release of the product. Updates occur at the end of each sprint. The amount of work to finish for a release is measured in story points, a special unit measuring complexity and risk of implementing a story. The timebox for a release is given as a number of sprints. If the trend line indicates the release finishing late the product owner may remove requirements with the customer.

Sometimes the product burndown is combined with a product burnup and workload fluctuation chart. The first shows the amount of story points already implemented while the latter shows when removed or added requirements changed the total workload which results in unexpected drops or jumps in the actual burndown. Figure 3.3 shows all named information combined into one chart (Pichler, 2008, P. 70ff).

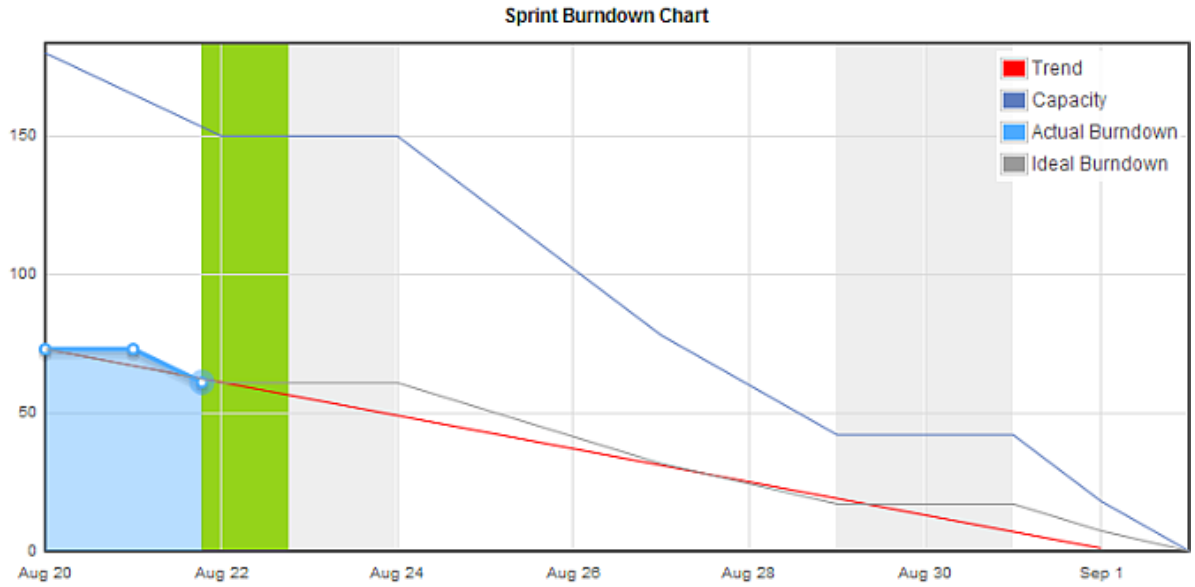

Figure 3.2: Sprint Burndown Chart

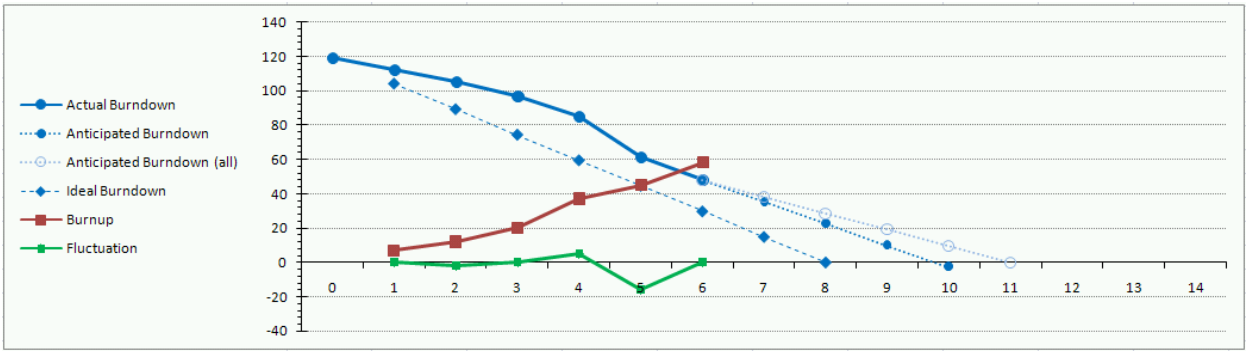

Figure 3.3: Release Burndown Chart

Product Increment

The product increment is the work result of each sprint and is defined as a partial, but working and deployable implementation of the final software product. In this definition, *working* refers to the requirement the product increment has to be fully tested and defect free. For the functionality represented by the product increment tests on all test levels have to exist, for example unit tests, integration tests, black box test, etc. To be deployable, the functionality of the increment has to be implemented end-to-end through all layers of the software. Each increment builds on and extends the previous increment. Figure 3.4 depicts stated principles. The horizontal bars form software layers while the vertical bars show increments. (Pichler, 2008, P. 83f)

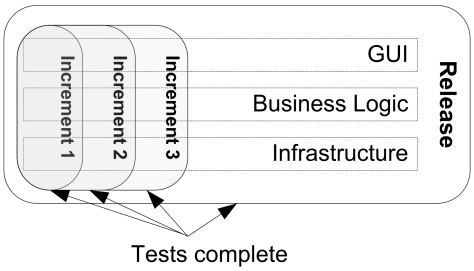

Figure 3.4: Product Increments, based on (Pichler, 2008, P. 84)

3.1.4 Meetings

Several meetings are recommended for a successful scrum project as listed below. All of them reoccur in more or less regularity during the project lifetime. The number of meetings necessary might look like an overhead at first, but it has to be considered project work such as planning, requirements analysis, design and so on usually done upfront in traditional approaches is done in those meetings and thereby spread over the whole project lifetime.

Each meeting is timeboxed and aborted if its timebox is used up. This approach limits wandering off subject, precludes fatigued or bored participants and motivates to compile and stick to an agenda. The scrum master facilitates all meetings.

Estimation Workshop. In the estimation workshops user stories and, if existent, epics are estimated. It is conducted several times before starting the first sprint and on demand during development if requirements are newly defined or changed. The product owner explains and the team estimates story after story.

To quickly establish a sound estimate Planning Poker (Cohn, 2006, P. 56ff) is played. After the story is explained each team member chooses an estimate from a predefined set of numbers. All team members then show their card simultaneously. Team members with outliers explain their reasons for their estimate. This is repeated until the team reaches consensus. Planning Poker is quick, transparent and a team decision. (Pichler, 2008, P. 61f)

Sprint Planning. Sprint Planning initiates each Sprint. First, a sprint goal is agreed on which guides the selection of stories and decisions made during a sprint. Second, a series of stories is selected limited by the average amount of story points the team completes each sprint. For each story, a number of tasks necessary to develop the story is defined and estimated. Finally, if the team is confident to be able to complete all stories it commits to complete them. Otherwise, the story selection needs to be modified.

To be effective, several things have to be kept in mind. The sprint planning is timeboxed therefore design discussions should be limited. The product owner is absolutely necessary to explain stories, but commitment, task definition and

planning is to discretion of the team. (Pichler, 2008, P. 87ff)

Daily Scrum. The Daily Scrum is a short (15 minutes) daily meeting in order to synchronize the team member's efforts to complete the sprint goal. Each team member answers three questions:

1. What did I do since the last daily scrum?
2. What do I plan to do until the next daily scrum?
3. What is blocking my progress?

The first two questions help to detect and deal with dependencies between tasks. The answers to the third question are impediments the scrum master has to act upon if not dealt with immediately. Additionally, the sprint burndown is examined in the daily scrum to monitor overall sprint progress. (Schwaber, 2007, P. 133)

Sprint Review. The sprint review concludes the sprint from the customer's perspective. The team presents the product increment the product owner, customer, management and other interested stakeholders. The sprint review is always a live demonstration of the product increment. Presentation slides and other artifacts, produced solely for the review, are disfavored. (Pichler, 2008, P. 107ff)

Sprint Retrospective. The sprint retrospective concludes the sprint from the team's perspective. The past biggest challenge or any other team or process concern is analyzed and improvements sketched out. One improvement is selected and fleshed out to be experimented with in the next sprint. Additionally, it is important to follow up the last sprint's experiment. (Pichler, 2008, P. 111ff)

It is not defined whether the review or the retrospective comes first. The author prefers to finish with the retrospective. The sprint review definitely marks the end of work for a sprint and customer feedback might be useful in the retrospective.

3.1.5 Pitfalls and Benefits

As mentioned at the outset of this section Scrum has a reputation of failing frequently. Often misconceptions are the reason. These and their possible root causes as identified in literature (Pichler, 2008; Highsmith, 2002) and known to the author are detailed below.

Scrum is complete. Scrum is a framework per definition. It provides an outline for a sound software development process. What is often ignored is, it needs to be supplemented with sound software development and project management practices. To accommodate ever changing requirements refactoring needs to occur. Refactoring needs to be backed by a rigorous test suite and integration policy. The

product mission may be more concise and significant if a goal definition method is used. Requirements still have to be elicited and priorities should not be just guesswork.

Scrum is easy. Since Scrum comprises only a few, concise rules it seems easy to be implemented. However, in most cases the ideological gap between the present methodology or process and Scrum is huge. Implementing Scrum requires a lot of change and will unearth unpleasant truths about the current situation. Consequently, hard work and perseverance is imperative.

Scrum is just new vocabulary. Sometimes a Scrum implementation is reduced to rename existing roles with the new Scrum vocabulary and proceed as usual. Reasons are the implementation is actually really difficult and just the tag, not the necessary effort, is welcomed. Additionally, some concepts may seem counterintuitive and are not fully understood without further learning and experimenting.

Scrum is a universal solution. Scrum is well known and wideley discussed. Often the decision to use it, is made because someone heard or read about it. Conditions and constraints are ignored and the implementation fails because the setting is not (yet) suitable for using Scrum. Nevertheless, since Scrum is a framework it provides an excellent baseline for a custom agile development process.

If Scrum is implemented and used accurately, a series of benefits are the reward. (Pichler, 2008)

Increased Customer Satisfaction. By focusing on what the customer really wants and prioritizing requirements accordingly, customers get solutions which really solve their problems. The results are high satisfaction and often even surprise. The challenge is, sometimes the customer needs to be disappointed deliberately by rejecting or removing some requirements in order to keep the product simple and not cluttered with features.

Increased Staff Satisfaction. A lot of responsibility is handed over to employees. Since software development is knowledge work usually highly skilled professionals work on a software product. They know how to solve given problems and take pride in finding creative solutions for challenges. By not dictating the exact procedure not only a vast knowledge pool is utilized but also employees feel more challenged and taken seriously. As a result, motivation and satisfaction increases.

More Stable and Faster Development. The development schedule as well as the software itself is stabilized through using sound development and planning methods. Reliable and truthful data about capabilities and progress is available for scheduling and controlling.

Increased Profitability. Through prioritized feature development based on customer value and costs the features posing high value can be put to productive and profitable use earlier and at the right time. Features low in value but high in cost are cut off at all.

In conclusion, a successful Scrum implementation might necessitate stepping out of the comfort zone but it is worth the effort.

3.2 eXtreme Programming (XP)

Extreme Programming assembles a set of well proven implementation and management techniques and orders to practice them as thoroughly as possible or, in other words, to take them to the extreme. Hence, it was named eXtreme Programming. From the vast number of practices known those where chosen which support each other best and fit the agile ideas. (Beck, 2000, P. XVff)

As basis for the practices and general behavior guidance XP names four values and five central principles derived from those values.

3.2.1 Values and Principles

The four values are (Beck, 2000, P. 29ff):

Communication. In many projects problems can be tracked down to important information was not communicated. In XP, the practices and an appointee are selected to keep communication flowing freely.

Simplicity. Today's solution must not solve tomorrow's problem since the latter probably will never exist.

Feedback. All artifacts are subjected to inspection. Thereby, feedback is gained on different time scales. Testing and early productive use of software give feedback about the code. Estimation, implementation and communication provide feedback about requirements and so on.

Courage. Without courage the other values cannot be taken to the extreme. Open, honest communication, throwing away or changing complex code and putting software into early production all needs courage.

Since values are vague and have different meanings for different people, the principles give a more concrete definition of how to interpret the values. Additionally, the principles are used to guide practice selection. (Beck, 2000, P. 37ff)

Rapid feedback reduces the time available to deviate and is essential for learning and reacting most appropriately.

Assume simplicity is possible for all solutions. *'The time you save on the 98% of problems for which this is true will give you ridiculous resources to apply to the other 2%.'* (Beck, 2000, P. 38)

Incremental Change is applied to all software development activities and enables easier detection of where errors actually occurred.

Embracing Change is a strategy that preserves the most options while solving the most pressing issue.

Quality Work is the only option in an agile environment.

Besides the five central principles ten other principle are listed to guide behavior in specific situations in collaboration, communication, mindset and software measurements.

3.2.2 Basic Activities, Strategies and Practices

XP names four activities as the basic activities of software development. First, coding to get something done. Second, testing to now when it is done. Third, listening, to know what to code and test and finally, designing to organize the system to support ongoing coding, testing and listening. (Beck, 2000, P. 37ff)

Besides listening, those activities are rather technology oriented. But in software development for a sustainable bussiness other activities such as planning have to get their share of attention as well. Therefore the practices are presented based on the strategies described in (Beck, 2000).

Planning Strategy

Plans are made for different time horizons. But the further in the future the horizon the lesser the details of the plan. Iteration, release and product comprising a few releases are the horizons of XP.

For planning decisions as well as all other decisions neither business nor technical considerations should be paramount. Hence, both business and technical people provide input for planning. The business side decides about scope, priorities and compositions and dates of releases. The technicians decide about implementation estimates, consequences of technology choices, development process and detailed scheduling. Both

parties use input from the other for their decisions and then jointly agree on a suitable plan. (Beck, 2000, P. 81ff)

The practices used in planning are detailed below.

Small Releases. Releases are planned based on business value. Each time valuable requirements are completely developed, they are released to the customer. This leads to release cycles of one or two months length.

Small releases mitigate the effect of errors in planning and design on the overall product since the errors are discovered early.

The Planning Game. The Release Planning Game brings business and technical people together. The goal is to find the maximum value of the software produced in a release's timeframe with low time investment.

First, the possible requirements are elicited and then a subset to be implemented is selected. The requirements and their details are written down as stories on physical cards, the story cards. Value, priority and implementation effort are determined. Finally, a subset is selected, based on which provide the most value and can be implemented with the given capacity and timeframe. The subset may be changed in consent if either development underestimated the effort or business overestimated value or discovered new, more valuable requirements.

The Iteration Planning Game prepares the stories for implementation. Programmers select stories based on the available capacity and write tasks which describe how to turn the stories into software. Tasks have to be smaller than stories. Each programmer accepts responsibility for a couple of task according to his abilities and availability. During implementation progress is measured about every second day to be able to react to overcommitment or underestimation.

The capacity for the release and iteration is limited by the average work completed during an iteration. XP uses ideal engineering days to estimate both stories and tasks. One ideal engineering day is a day of uninterrupted work. (Beck, 2000, P. 86ff)

Design Strategy

A lot of complexity in software products stems from solutions for anticipated problems. If the problem never arises, but the solution is already in the product, it is very difficult to remove it again. The XP design approach is to design in very small steps for today's problem. Design and write a test case. Design and write the simplest code to get the test running and repeat. If it is difficult to add new design, refactor.

Design through refactoring works if the system is constantly kept at the simplest possible level. Each time new code is written existing code is cleaned up and made simpler

as well. A little more effort than for the current task is invested each time in order to avoid getting stuck in future. This refers to cleaning up code only, not to prepare for anticipated features. If bigger changes are needed they are completed in small steps towards a set goal as well. (Beck, 2000, P. 103ff)

Doing the simplest thing is not easy it is sometimes even harder than a more complex solution. The XP definition of simplest guides the design process. In priority order simplest means (Beck, 2000, P. 109) the system (code and tests together)

- must communicate everything you want it to communicate.
- must not contain duplicate code.
- should have the fewest possible classes.
- should have the fewest possible methods.

Three practices are used in the design process.

Metaphor. The metaphor is a simple shared story about what the system should comprise. It is used for guiding development, communicating about the system and naming technical entities. The metaphor also captures part of the system architecture.

Simple Design. Simple design is achieved by adding everything you need for today and removing everything you do not need. The design then has to fulfill four simple conditions. First, it runs all tests. Second, no duplicate logic or code exists. Third, it states every intention important to the programmers. And finally, it has the fewest possible classes and methods.

Refactoring. The system is restructured without changing its behavior. Refactoring is not done on speculation, but if the system does not follow the definition of simple anymore.

Development Strategy

The development strategy is best described as programming augmented with five supporting practices. (Beck, 2000)

Testing. Unit and functional tests compose the core of XP testing. Unit tests are written by programmers and kept in sync with the code. They are written before writing new code or refactoring, for those parts which might break, represent unusual usages or already caused problems. Unit tests have to run at 100%. No other code is written before a failing unit test is fixed. Functional tests are written

by customers to define and test when a story is done. They do not need to run at, but should approach 100% over time.

Those kinds of tests are supported by others as needed. For example, stress tests, performance tests, monkey tests and others are used as required.

Continuous Integration. After completing a task the changes are integrated with the whole software. Collisions with other changes and failing tests are fixed. Eventual issues or bugs are at most a day old and therefore easier to find and fix than some created a few weeks ago.

Collective Ownership. At any time any programmer can change any piece of code. Tests and continuous integration ensure the system is not broken completely. Collective ownership spreads the knowledge of the system around the team and empowers each programmer to work without depending on someone else. Additionally, complex code is removed early or even precluded at all.

Pair Programming. Two programmers work at one machine to complete a task. While one partner implements the code, the other thinks about the overall approach, test cases, possible simplifications and refactoring. The roles are switched at the discretion of the pair. Pair programming enhances communication skills and code quality, prevents from skipping practices, speeds up information exchange and knowledge creation.

Coding Standards. The code has to look like written by one person, otherwise changing between different parts of code, refactoring or pair programming is very cumbersome.

Collaboration Strategy

XP is build on broadband communication and tight collaboration. Programmers, designers, architects, technical writers and all staff needed to build a system are collocated in a work area. The work area should include public space for programming, private space for undisturbed work or phone calls and a little communal space to chat, eat, drink and rest.

The collaboration strategy also includes two other practices.

On-site Customer. A customer representative, who will actually use the system, is collocated with the team. He is available full-time to answer questions and provide further input.

40-hour Week. Creative problem solving needs fresh minds. Work done tired usually needs to be repeated or reworked. Hence, no one works more than 40 hours a week and no second week of overtime in a row.

3.2.3 Roles

Working in teams does not remove the need to assign responsibilities. XP suggests a couple of roles to shoulder certain responsibilities. Roles can be shared or combined into one person. (Beck, 2000, P. 139ff)

Programmer. Programmers design, write the code and tests and refactor. But in contrast to other software development disciplines code is not only written to develop features but also to communicate. Names, structures and other code artifacts need to reflect intent. If they do not, refactoring is required. And, of course, programmers need to communicate with other programmers and the customer.

Customer. While the programmer knows *how*, the customer needs to know *what* to program. He writes the stories, develops functional tests for the stories and decides about value and priorities. It is vital for the customer to be able to really make those decisions with confidence but also to change them if necessary. An actual user of the system in development but with a broader business view is the best choice for an on-site customer.

Tester. The tester is responsible for the functional tests provided by the customer. He supports the customer in writing the tests, runs the tests regularly if not automated, broadcasts test results and maintains the test tools.

Tracker. To improve a system or the process it is sensible to collect data to suggest and measure improvements. The tracker collects data about estimations, progress, defects, test scores and other metrics chosen without interrupting the progress too much. Additionally, he needs to be able to tell if set commitments are jeopardized.

Coach. The coach is responsible for the process and its technical outputs. He corrects process deviations and subtly points out issues. The coach's foremost goal is to make his role redundant by teaching the team everything it needs to take over the responsibility conjointly.

Consultant. Working together on all parts of the system produces more generalists then specialists. Hence, sometimes external experts have to be brought in to help out with deep technical knowledge. The role of the consultant is to teach the team and enable them to solve a similar problem by themselves next time.

Big Boss. To know if things go awry, the big boss has to allow open and honest communication to give himself enough time to react. Interventions should be used rarely and only if the team really screws up.

3.3 Lean Software Development (LSD)

Lean Software Development provides a whole new perspective on agile development. Many principles and practices widely adopted are explained from the viewpoint of management in terms of costs. In addition, strategies, how to measure and optimize software development processes are given. That is why LSD not only helps to optimize software development but also fosters agile adoption processes by creating additional support from management and development. In the authors opinion, LSD provides essential explanations why to use certain agile practices.

Lean Software Development is not be confused with Lean Development (LD). LSD originated 2003 from the first book written Tom and Mary Poppendieck (Poppendieck and Poppendieck, 2007, P. XXIII) and LD was developed by Bob Charette (Highsmith, 2002, P. 285). Both stem from lean production, but LSD formulates seven principles while LD names twelve principles derived from four success factors to fit the lean idea for software development.

3.3.1 The Idea

Flourishing organizations such as Toyota (Womack et al., 2007), Dell or 3M (Poppendieck and Poppendieck, 2007) apply lean management approaches successfully since decades. The basic idea behind Lean Software Development is to transfer the well proven lean principles from manufacturing or logistics to software development.

In short, being lean means to create true value, move fast, respect people and partners while producing as little waste as possible. Lean management often feels counterintuitive because it breaks with many long learned habits thought intrinsic for successfully managing an organization. For example, truly lean organizations work with their partners as if organizational borders would not exist. Innovations and improvements are shared with the whole value chain to optimize the whole, not just a specific organization.

3.3.2 Principles

This idea can be adopted by adhering to the seven principles of lean software development.

Principle 1: Eliminate Waste

The goal of eliminating waste is to look at the software creation timeline and remove each and every nonvalue-adding waste. The timeline may be different for different organization but it should start as early and ends as late in the creation process as possible, e.g. from receiving an order to deploying the software to the customer.

Since waste is anything that does not add value, it is crucial to know what value is. In software production value is usually defined by what customers want. But customers tend not to be able to know or articulate what they really want. Therefore, it is important to gain deep customer understanding and learn their true needs by using concepts such as the Kano Model, piloting, customer focus groups, chief engineers and more (Poppendieck and Poppendieck, 2007, P. 43ff).

The next step is to learn what waste is. LSD names seven wastes of software development translated from the seven wastes of manufacturing (Poppendieck and Poppendieck, 2007, P. 73ff).

Partially Done Work. *'The inventory of software development is partially done work.'* (Poppendieck and Poppendieck, 2007, P. 75)

Inventory items tie up money, hide quality problems, get lost or grow obsolete. Typical examples of partially done work are uncoded documentation such as defect, request and specification queues, unsynchronized code, untested code, undocumented code and undeployed code.

Extra Features. The overproduction of software development are extra features that are not needed to get the customer's current job done. They just add to cost and complexity and to make things worse, keep doing so forever, since they cannot just be taken off the shelf and thrown away like an overproduction inventory in manufacturing.

Relearning. Relearning occurs when decisions made and things learned during development are not captured and preserved for future reference and therefore have to be repeated. Knowledge is also wasted if the knowledge and experience of employees is ignored and not drawn on.

Handoffs. Handoffs occur when work requiring tacit knowledge for execution is passed on to colleagues or other teams. The majority of tacit knowledge cannot be transferred with the work and therefore is wasted. After three handoffs only twelve percent of the knowledge is left, based on an very conservatively estimated transfer rate of 50%.

Task Switching. Software development is knowledge work, therefore each time a new task is started, a certain amount of time needs to be invested to understand the problem at hand and find the correct solution. Every time one needs to switch

to a different task, the time to get started needs to be invested again. Additionally, the value a task could deliver earlier, if finished in one go, is wasted.

Delays. Waiting for people or resources to be available causes delay. Interdisciplinary, collocated teams and regular feedback through short iterations and fast hardware infrastructure help to decrease delays and increase quality of decisions.

Defects. Finding defects in the verification phase should be the exception not the routine. Otherwise the process itself is defective because not enough time is spent on mistake-proofing the code in the first place.

Finally, waste needs to be discovered and then removed, biggest first. The Poppendiecks (Poppendieck and Poppendieck, 2007, P. 83ff) introduce value stream maps as their tool of choice for exposing waste. In a value stream map all activities of a certain process are mapped with their average completion times and delays between the activities. An efficiency measure is then computed by dividing the time actual working on the activities through total process time. Value stream maps also prevent local optimization (cf. Principle 7: Optimize the Whole) since they include the whole process if used correctly.

Figure 3.5 shows a value stream map for a high-priority feature change request process. In this example the two hour wait for technical assessment is waste if the developer could do the assessment (Poppendieck and Poppendieck, 2007, P. 86).

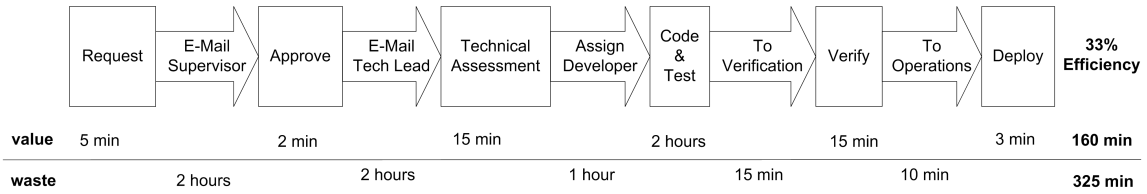

Figure 3.5: A value stream map for a high-priority feature change request (Poppendieck and Poppendieck, 2007, P. 86)

Summarized, the steps in eliminating waste are, learn what value is, learn what waste is, define the timeline and reduce the timeline by removing all nonvalue-added wastes.

Principle 2: Build Quality In

Building quality in requires controlling conditions so they do not allow defects to occur. Inspections to prevent defects are valuable while inspections to find defects are often waste as are the resulting long defect queues. Quality needs to be build in from the start not tested in.

Mistake-proofing processes and products is a method to build quality in. Standards for tools, names and code prevent misusage. Automation of repetitive tasks precludes

missed steps and frees resources for more complex tasks. Test-Driven Development prevents defects to occur (again).

In software development often a compromise is made between speed and quality, but with excellent, rapid feedback and superb discipline both can be achieved. The concepts for rapid feedback are short iterations, release minimum feature sets incrementally, review code and software and plenty of customer communication. Practices for technical feedback are continuous integration and testing on different levels. (Poppendieck and Poppendieck, 2007, P. 177ff)

Discipline unfortunately is not that easy. It is not achievable just by introducing new practices. It is the effort put into really using and keeping the practices. The Five S's (Poppendieck and Poppendieck, 2007, P. 190) offer a good start to achieve discipline.

1. **Sort.** Sort through file storages, code bases, office storages, procedures and regulations and remove everything that is not needed or not used any more.
2. **Systematize.** Find a logically organization for file structure, cabinets and workplaces. Everyone needs to be able to find things quickly and in the same place as always.
3. **Shine.** Clean up garbage and get ready for work.
4. **Standardize.** Put standards and eventually automation in place. Make clear for everyone how things are expected to be.
5. **Sustain.** Keep up the discipline to follow the standards and keep everything clean. (Martin, 2008, P. 14) suggest to apply the Boy Scout rule 'Leave the campground cleaner than you found it.' to sustain clean code. Each time the code is returned to the main code base it has to be a little cleaner.

In conclusion, the indispensible foundation of *quality build in* is a sound process used disciplined.

Principle 3: Create Knowledge

In software development rarely all necessary knowledge is available from the very beginning. Customers are not able to articulate what they really need. New technology or hardware behaves differently than expected or documented. The market does not react to new features as expected. Consequently, software development is a knowledge creating process.

The ability to create knowledge must be built into the process. Knowledge should not only be a by-product of writing software. A disciplined approach to problem solving is recommended. Such an approach includes the steps problem definition, situation

analysis, hypothesis creation, experimenting, result verification and standardization (Poppendieck and Poppendieck, 2007, P. 169).

As important, if not even more important than to create knowledge is putting knowledge to a good use. With a process that requires locking down specification, plan and design upfront one cannot expect to utilize created knowledge to the full extend. Another principle is to defer decisions to the last possible moment. It enables to use the maximum amount of knowledge and raises decision quality (cf. Principle 4: Defer Commitment).

Finally, knowledge needs to be preserved. Most knowledge in software development is tacit knowledge, therefore it is difficult to make it available in written form. Used methods are to let employees write engineer diaries and concise reports. Such a concise report is the A3-report which requires capturing all necessary details about a customer job, a decision or a problem on an A3-page. (Poppendieck and Poppendieck, 2007, P. 149ff)

Principle 4: Defer Commitment

Deferring commitment requires waiting for the last possible moment to make an irreversible decision because then the most information is available. Consequently, it is necessary to learn how long one can wait before it is too late.

Moreover, most of the decisions should be made reversible in order to easily change them later. Specifications are developed in concurrence with the software. Early developed features do not lock down design and architecture. Dependencies are broken so features can be added in any order. Options are maintained by developing multiple solutions. (Poppendieck and Poppendieck, 2007, P. 32f, 244f)

One approach for developing multiple solutions is set-based design (Poppendieck and Poppendieck, 2007, P. 160ff). In set-based design multiple solutions are developed in order to have the best solution available at the right time. This seems really counter-intuitive and wasteful because it can be seen as developing a couple of solutions and throwing them all, except one, away. But it actually enables having a suboptimal solution ready to use early while concurrently developing one or more better solutions for later use. This is best used for high-impact irreversible decisions or almost impossible deadlines.

Principle 5: Deliver Fast

Delivering fast is essential in a rapidly changing business such as software development. Slow delivery allows customer or market to change their needs before the soft-

ware is ready. Fast delivery, if paired with superb quality and deep customer understanding, is also a market advantage difficult to copy.

The crux is, speed is not equal to hacking. True, sustainable high speed can only be achieved on basis of superb quality because complex, unstable code inevitably slows down. In lean software development speed is achieved by removing waste, not quality from core business-processes. The fundamental lean equation is, less waste frees resources to create value and allows delivering faster.

In stable systems, the average amount of time to get something through a process (cycle time) is limited by Little's Law (Equation 3.1). Thus, cycle time is reduced by either increasing the average number of items completed or decreasing the number of items in process.

$$Cyle\ Time = \frac{Number\ of\ Items\ in\ Process}{Average\ Number\ of\ Items\ completed} \tag{3.1}$$

To draw on that simple law, the system needs to be maintained stable through reducing variation and keeping an eye on utilization. Variation is how much items in a batch differ from each other. The smaller the batch is, the smaller is the variation. For example, in 60 minutes of work less things can go wrong than in six weeks. Utilization is the percentage of the used capacity of a certain resource. It is established that utilization above 80% tends to slow everything down. Thus, it is not beneficial to schedule 100% of available hours for projects. (Poppendieck and Poppendieck, 2007, P. 100ff)

Once the system is stabilized, cycle time can be reduced by drawing on proven strategies.

Even out the arrival of work. Queues of work at the beginning of the development process allow releasing the work to development at an even pace. Nevertheless, the queues should not be unnecessary long otherwise development will be out of sync with customer needs. Therefore, the queues should also be filled at a steady rate rather than in big batches once or twice a year.

Minimize the number of things in process. As mentioned, unfinished work is the inventory of software development with all its evils. Less obvious is, customer requests waiting for approval are, from the customer's perspective, unfinished work. Every discussion, estimation and prioritization of the customer request queue consumes resources. Therefore, request queues should be kept reasonable short by rejecting requests which will not be completed anyway.

Minimize size of things in process. The size of things in process is a function of the release cycle length or work package size. Both should be kept short and small.

Establish a regular cadence. Iterations establish a cadence. At the end of an iteration about the same amount of work is done. Plans and commitments to customers can be made with confidence. Iterations should be short enough, so that there is no flurry at the end and customer changes can wait until the end of the iteration.

Limit work to capacity. Overtime is not sustainable. Schedules have to be limited to capacity with an eye on utilizations.

Use pull scheduling. Pull scheduling allows developers to choose the amount of work they are confident to complete in one iteration. Work is pulled from the top of a prioritized queue of work, and scheduled work is limited to capacity automatically.

Little's Law as well as listed strategies are drawn from queuing theory which is applicable since queues are very common in software development. Queues exist for requirements, defects, change requests and more. Even the work planned for an iteration is a queue. (Poppendieck and Poppendieck, 2007, P. 100ff)

Principle 6: Respect People

Respect for people is the core principle of lean management. Trust and care for the people creates an engaged, creative and thinking workforce. Respect for people is built on three cornerstones (Poppendieck and Poppendieck, 2007, P. 37):

Entrepreneurial Leadership. Almost every highly successful project can be traced back to excellent leadership on creating a great product. Developing and respecting good leadership which fosters engaged, thinking people is essential.

The leader also needs to have a deep understanding of market needs and the technology. Together with the team the technology is used to meet the need in a unique way. The leader makes subtle corrections to synchronize efforts to create a great product.

Expert Technical Workforce. A company that buys all expertise has no competitive advantage since bought expertise is available for everyone else too. Thus, appropriate technical expertise has to be developed and nurtured inside the organization. Teams need to be staffed with all technical expertise necessary to achieve their goals.

Expertise is developed by developing a challenging work environment and coaching. Seasoned engineers need to be available to mentor newcomers. Promoting them into management levels to early deprives the workforce of a vast amount of tacit knowledge. Goals are set achievable with the available expertise and the combined team effort.

Responsibility-Based Planning and Control. General Plans and goals are given to the people and they are trusted to figure out what to do and how to achieve those goals. After all, experts are trained to do just that. Teams are formed and responsibility is handed over to plan and control their work by themselves.

Self-organization is supported best by a visual workplace. Dashboards and signals are used to display plans, goals, designs and states of various important artifacts. That way, important information is immediately available for reference and important events are not missed.

One of the easiest ways to undermine respect and team work is to use monetary bonuses and incentives if based on individual performance. Employees almost immediately stop helping each other and solely concentrate on figuring out, how their performance will look best. Therefore, at best motivation should not be achieved through money and performance ratings used for employee development only, not annual raises.

Principle 7: Optimize the Whole

The lean principles are very much interlinked. It should be obvious, local optimization is almost impossible if whole processes are optimized. But lean management takes global optimization even one step further by optimizing beyond organizational borders.

Partnerships and global networks are formed to create synergies. Partners and networks operate as if they belong to the same organization. All companies inside the network share the same management system (lean management) and optimizations are planned for the whole network. This approach does not only foster understanding and collaboration, but also achieves a cost advantage of 25-30% over competitors (Poppendieck and Poppendieck, 2007, P. 39).

3.4 Goal Directed Project Management (GDPM)

Goal Directed Project Management (GDPM) is developed and refined in Norway since 1970 (Andersen et al., 2004, P. v). Its distinctive features are the strong focus on project goals rather than detailed plans as project foundation, and the belief projects have to be viewed from a broader than just the technical perspective. The term PSO (people, system and organization) project was coined to emphasize people and organizations have to be developed along with technical system development.

In GDPM, project management is partitioned into three levels and three main tasks. Table 3.1 shows the recommended tools for each level and task combination.

Task / Level	Planning	Organization	Control
Foundation of Project	Project Mandate Mission breakdown structure Stakeholder analysis	Principle responsibility chart	
Global Level (milestone level)	Milestone plan Uncertainty analysis	Milestone responsibility chart	Milestone report
Detail level (activity level)	Activity responsibility chart		Activity report

Table 3.1: Overview of Goal Directed Project Management (Andersen et al., 2004, P. 2)

As depicted, three different levels of project work are distinguished. Both, project plan and project organization are planned on different levels of detail. This prevents changes or unexpected events from throwing over the whole project outline. GDPM also facilitates intermediate deliveries on the global level rather than *big day* deliveries at the end of a project (Andersen et al., 2004, P.14). The following sections discuss the different levels in more detail.

3.4.1 Project Foundation

The foundation is of great importance in GDPM. It sets purpose and goals, project boundaries and limitations and principle organizational responsibility arrangements.

Purpose and Goal

Project foundation work starts with the clarification of the project's purpose and goals. The main tool is the mission breakdown structure and a goal hierachy defined jointly by project owner, project manager and other necessary stakeholders.

The purpose of a project expresses a future desired situation for the organization receiving the project deliveries. Hence, the purpose of a project must be aligned with the company's purpose or its support or management functions. To begin with, a main purpose is formulated which is then further broken down into more detailed purposes covering the PSO parts. Thus, a mission breakdown structure is build up as shown in figure 3.6. Using the mission breakdown structure, the project scope may be limited to

certain detailed purposes.

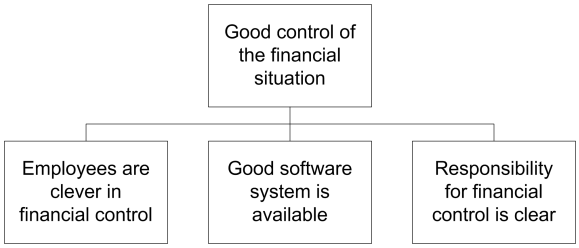

Figure 3.6: Mission breakdown structure of an example project, extract from (Andersen et al., 2004, P. 44)

The goals are then derived from the selected purposes. Again, several main goals are detailed with several subgoals. The difference is, a purpose expresses a future desired situation and a goal specifies a measurable project delivery. Furthermore, the subgoals should be specified to induce several deliverables during the lifetime of the project. (Andersen et al., 2004, P. 38ff)

Stakeholder

Stakeholders are individuals or groups who are either actively involved in, or positively or negatively affected by the project. The stakeholders have to be identified to understand their motivations, collaboration and contributions, establish information distribution and common purposes and goals.

GDPM suggests using a matrix for stakeholder analysis listing the stakeholders top down and the most important factors left to right. Important factors from the stakeholder perspective are the area of interest, contributions, expectations, power over the project, strategy to work with the stakeholder and person responsible for executing the strategy. (Andersen et al., 2004, P. 48ff)

Principle Responsibilities

Most projects are embedded in a base organization. The base organization executes day-to-day tasks while a project is set up for a specific endeavor. Since projects have to draw from and share resources with the base organization, conflicts may arise. Hence, responsibilities and roles have to be clarified between the base organization and a project.

GDPM uses the principle responsibility chart to divide responsibilities between the project and the base organization. The elements of a responsibility chart are roles from left to right and organizational or project matters top down. Symbols (Table 3.3) define

the responsibilities. Table 3.2 shows a few lines of an example for a principle responsibility chart. The chart is defined once and used and adjusted for all projects in the organization.

	Managing Director	Project owner	[..]	Project manager	Implementer	[..]
Principle responsibility chart	C	C		P|X		
[..]						
Milestone report		D		P|X	C	
Activity planning		C		P|X	C	
[..]						

Table 3.2: Example of principle responsibility chart, extract from (Andersen et al., 2004, P. 58)

X	Executes the work
D	Takes decisions solely or ultimately
d	Takes decisions jointly or partly
P	Manages work and controls progress
T	Provides tuition on the job
C	Must be consulted
I	Must be informed
A	Available to advise

Table 3.3: Responsibilities in a project with their abbreviations (Andersen et al., 2004, P. 105)

Project Mandate

The project mandate summarizes the different parts of the project foundation. It comprises several elements.

- **Name of the project.** The project's name should reflect the projects purpose.
- **Project owner.** Has the main responsibility for the project on behalf of the organization receiving project deliveries.

- **Project background.** Relevant history and background information give a better understanding of the project.
- **Purpose of the project.** Mission breakdown structure.
- **Goals of the project.** Goal hierarchy.
- **Project scope.** Includes and excludes concerns from the project.
- **Limitations on project work.** Time, cost and other limits, unacceptable solutions or actions restricting the project.
- **Project budget.** Depicts the financial situation of the project (e.g. costs, profit) concisely.

The project mandate is used to decide whether to start a project and serves as the main point of reference for all aspects of a project. (Andersen et al., 2004, P. 51ff)

3.4.2 Milestone Level

The milestone plan structures the project on the global level. Milestones serve as checkpoints to be achieved on a certain date. Milestone responsibilities clarify roles and an uncertainty analysis reveals project risks. The milestone report shows progress in relation to the milestone plan.

Milestone Plan

The milestone plan combines several project aspects into one logical structure. Project work is organized along result paths and deadlines are set for each milestone. The milestones themselves are organized in a logical sequence and responsibilities for each milestone are defined.

First, milestones for a project are defined. A milestone describes what state the project should be in at a certain time. Hence, milestones are not specified as activities and should not suggest or ask for a certain solution.

Once the milestones are determined, the dependencies between them are defined. A dependency states a milestone cannot be completed before its preceding milestone. However, work for a milestone can be started without dependency on other milestones. Milestones and their dependencies are displayed as in figure 3.7. The different letters in figure 3.7 distinguish different result paths. A result path comprises closely related milestones contributing to a common result. Usually the number of goals is reflected in the number of result paths. (Andersen et al., 2004, P. 76ff)

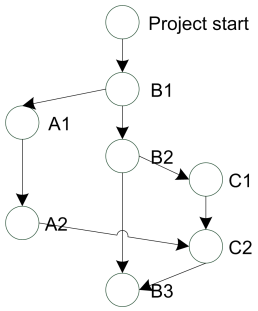

Figure 3.7: Example for a milestone plan

With the completed milestone plan a project schedule is drawn up. For each milestone the most time- and resource-consuming activities are identified. A deadline for a milestone is set based on resource availabilities. Additionally, for each milestone responsibilities are defined, similar to the principle responsibility chart. Table 3.4 shows a few lines of a milestone schedule with responsibilities.

Man-days	Start	Week	End	No.	Milestone	Project manager	Managing director	[..]
10	1/1	1-4	20/1	M1	Description of present situation	X|P	A	
15	10/1	3-8	10/2	M2	Description of desired situation	X|P	D	
[..]								

Table 3.4: Example of milestone time schedule, extract from (Andersen et al., 2004, P. 118)

Uncertainty Analysis

Uncertainties are factors which may prevent the project from achieving its goals. To be prepared if uncertainties occur, an uncertainty analysis is carried out for both the project in general and each milestone in specific. Important areas to look for uncertainties are resources, environment, project commitment, project plans and organization, and points of decisions.

The results of the uncertainty analysis are recorded in an uncertainty matrix. Uncertainties are listed top down and factors left to right. Factors to be considered are the affected milestone, probability of occurrence, consequence, action and reaction, and person responsible. (Andersen et al., 2004, P. 120ff)

Milestone Report

The milestone report contains a concise statement about each milestone's state. More detailed reporting during lifetime of a milestone is done in the activity report. The latter is part of the detail level.

Generally, reporting activities should not end in themselves. Hence, criteria to report on are defined in advance. Criteria assessment is given concise and undisguised. Reporting is done on the original plan documents, not on separate report forms. If answers indicate deviations, following up with discussions and activities is paramount. Following those simple guidelines avoids wasting resources during regular project progress.

3.4.3 Detail Level

Detailed planning is part of ongoing project work. Detailed plans are not made in advance for the whole project, but for each milestone when it is about to begin. Those who execute the project work are included in detailed planning.

Activity plan

For each milestone all activities necessary to complete a milestone are identified. Next, affected people are identified and responsibilities assigned using the responsibility chart format. Finally, the effort for activity execution and the work duration in calendar time are estimated and determined.

Activity report

The activity report is used for immediate project control. Criteria to report on and control are use of resources, time schedule, quality, responsibility chart, changes/additions, waiting time and special problems. Whether a criterion is consistent with its target state is answered with a simple Yes or No. Table 3.5 shows an activity report as part of the activity plan.

3.5 Change Project Management

Change projects are probably the most challenging of all projects. They are conducted if a problem or opportunity necessitates permanent internal changes in an organiza-

Work Done	Work to do	On schedule	Quality accepted	Responsibility chart kept	Changes required	Waiting time	Special problems	Delay	Accumulated	Days	Start	Finish	Activity	Eve (Project manager)	Paul (Sales)	[..]
18,0		Y	Y							14,0	21/1	27/1	Map work processes sales	X	X|P	
6,0		Y	Y							4,0	25/1	27/1	Map processes purchasing	X		
[..]																

Table 3.5: Example of activity plan with report, extract from (Andersen et al., 2004, P. 222)

tion. Internal structures and processes are adapted to stay innovative and flexible.

But to be successful, a change project manager needs to be aware, there is more to a change project than just defining new structures and processes. In order to work with the new organizational measures, people need to change their habits. Change projects require to let go of beliefs, feelings, patterns, ideals and relations people had come to love.

Hence, a change project needs to operate on two levels (Lüsch and Zitzke, 2004, P. 17).

- **tangible level.** Structures, processes, forms of organization, etc.
- **intangible, mental level.** Social networks, tradition, culture, beliefs, identity, etc.

To achieve permanent and effective turnaround, a change project needs to transform both levels as shown in figure 3.8. Roughly speaking, about 70% of the total effort needs to be invested into the intangible level. This is telling just how important it is not to neglect the intangible aspects.

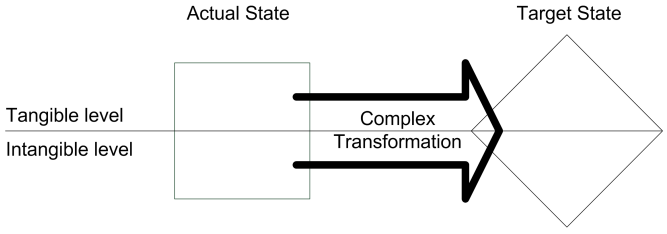

Figure 3.8: Complex Change Process (Lüsch and Zitzke, 2004, P. 17)

Figure 3.9 illustrates the consequences of reducing the change project to the tangible level based on the believe, the intangible level will follow once the tangible aspects are changed. People go into opposition and do everything to prove the suggested change is not working. Once the project is closed, everyone will go back to 'the old way'.

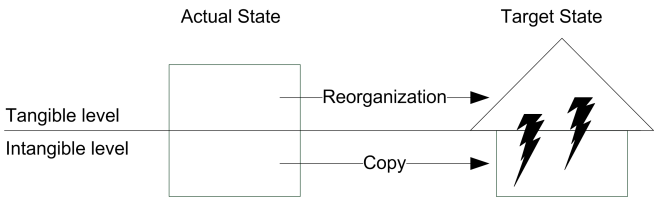

Figure 3.9: Reduced Change Process (Lüsch and Zitzke, 2004, P. 18)

Unfortunately, an approach to guaranteed success does not exist. Nevertheless, a number of special methods or attitudes are suggested to heighten the probability of success of a change project.

3.5.1 Basic Orientation

The foundation of each project is the definition of the orientation or goal. An accurate formulation of the goal facilitates combining and streamlining all efforts. The principle two antipodal orientations are shown in figure 3.10.

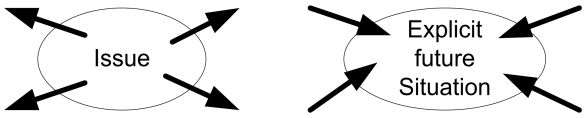

Figure 3.10: Two antipodal orientations in change projects (Lüsch and Zitzke, 2004, P. 21f)

The left side depicts a situation where the goal is formulated to get away from the issue. This allows forces to act uncoordinated and may result in objectionable situations. For example, if high costs of material are the issue, a possible solution and probably the easiest is to buy cheaper material. But in turn, this may cause customers to reject products because quality suffered.

A better guidance for all involved parties is provided by a goal formulated as explicit future situation. Energies are combined and directed towards the goal as shown by the right side of figure 3.10. Synergies result. Hence, the second orientation minimizes uncoordinated actions and heightens the success probability of change projects.

3.5.2 Forces

In organizations usually two antipodal forces are at work in an equal measure (Lüsch and Zitzke, 2004, P. 23).

- **Drawing forces** are attracted by the new situation and view the current situation as ineffective. They are willing to invest in change.

- **Impeding forces** experience the current situation as stable, safe and to be protected. The new situation seems threatening.

Once a change project is initiated, the two forces start to compete against each other. Unfortunately, they tend to try to balance each other out. The more the drawing forces try to prevail, the more resistance is built up by impeding forces. The consequence for the project manager is, he needs to keep both forces low and balanced. As a direct result, the overall energy investment is lower.

Although it is tempting to boost drawing forces, since they point towards the goal, the impeding forces need to be integrated. First and foremost this means not to dismiss, but to take them seriously by following certain ground rules (Lüsch and Zitzke, 2004, P. 24).

- Do not dismiss traditions disrespectfully.
- Value the past and past approaches as useful and necessary for the past situation.
- Permit disappointment, anger and regret.
- Regard objections and do not ignore or dismiss them.

An organization is usually a very factual environment therefore rule three is probably the most difficult to follow.

3.5.3 Course of Action

Based on the elaborations above a special course of action is suggested for change projects in addition to conventional project work. (Lüsch and Zitzke, 2004, P. 24ff)

Define goals conjointly

Since in a change project the number of affected parties is usually high it is not enough to leave goal definition to the project team only. Affected parties or selective representatives should be able to express their concerns, comment and amend goals.

A workshop is used to first present and then discuss the goals. The workshop's goal is to optimize the direction of change conjointly, not to decide whether to carry it out. Hence, an explicit principle orientation has to exist beforehand. (Lüsch and Zitzke, 2004, P. 24f)

Communicate open and effective

Once it is thought about change, it is inevitable information circulates through the grapevines and hopes, fears and rumors start to build up. The project manager needs to counteract with open, early and effective communication.

Four ingredients to an effective communication concept are listed below (Lüsch and Zitzke, 2004, P. 25ff).

Management initializes information. Complex change needs support from top management. By initializing the information process through management the commitment to support is made at the same time.

Start to inform early, but not too early. More time to acquaint oneself with change fosters a feeling of been taken seriously and stimulates to assume responsibility. On the downside, it allows more time for building up rumors and resistance if communication is unidirectional.

To inform means to answer questions. An excellent weapon against resistance is to openly answer questions and exemplify the necessity of change.

Inform face to face. Information, especially initial information, should happen face to face. Questions can be raised and dealt with immediately to leave little room for speculations or rumors.

In short, the four important questions to answer are who, when, what and how.

Utilize dramaturgy

It may sound strange, but for a change projects it is beneficial if it is supported by orchestrations for important activities. Reasons are the host of emotions and that more than half of the efforts need to be focused on the intangible level.

Orchestrations draw on existing emotions by using known symbols and rituals to bring about identity. Well chosen dates and locations convey certain traditions or values. Speakers, wording and supporting program of speeches evoke advantageous reactions and people feel emotionally involved. (Lüsch and Zitzke, 2004, P. 26f)

Simulate and test change

Even with the most meticulous planning important parameters can be overlooked since the change process is complex. To avoid problems and conflicts later on, the desired change should be simulated or experimented with.

In a simulation the new processes are theoretically run through step by step based on a fictitious or known future situation. More complex changes may be tested in an experiment. The changes are brought to bear in a delimited area of the organization. Reliable measures are kept while others are gradually optimized. For both, simulation and experiment, it is important not to protract the decision, otherwise change progress and effort may end in talk. (Lüsch and Zitzke, 2004, P. 27f)

Chapter 4

Adoption Process

The adoption of agile software development and transition from a project-centered to a product-centered software development was divided in three consecutive phases. First, the situation at hand was analyzed, second, an experiment to test the new processes was conducted and finally a transformation project was planned and initialized. The result of the transformation project is a complete organization manual for XiTrust.

4.1 Initial Situation

Prior to this thesis, the author spent about two years at XiTrust developing two other papers (Keuschnigg, 2007; Richter, 2009). Therefore the analysis of the initial situation is mostly based on personal experience with processes and organizational choices at XiTrust. To deepen the understanding, interviews on special topics with selected employees where used. The findings are grouped into organizational characteristics, types of projects, and areas to be improved.

The analysis and following description of organizational characteristics exhibited by XiTrust is supplemented by models provided in organization theory. Thus, XiTrust is

an Adhocracy. *'The adhocracy is a decentralised form which is characterised by high horizontal differentiation, low vertical differentiation, low formalisation, intensive coordination and great flexibility and responsiveness.* (Robbins and Barnwell, 2002, P. 122)

Typically almost all employees in an adhocracy are highly skilled professionals (horizontal differentiation) who work together to innovate, solve unique problems and perform flexible activities. The distinction between supervisor and employee is blurred and consequently levels of administration hardly exist (vertical differentiation).

The execution of work is based on skill and knowledge rather than rules and

regulations (formalization). Coordination is either through mutual adjustment or an employee taking on the role of a coordinator. (Robbins and Barnwell, 2002, P. 122ff)

It is common for small, young organizations like XiTrust to exhibit the characteristics of an adhocracy. The resulting flexibility and responsiveness is a big market-advantage often sought after and tried to be reproduced by larger organizations using adhocracy-like structures (e.g. Sponsored Spin-offs, Venture Management, (Haberfellner, 2008, P. 157)).

project-oriented. Almost all internal and external undertakings are managed as projects. Furthermore XiTrust exhibits the following characteristics presented based on the 7S-Modell (Haberfellner, 2008, P. 26ff).

- **Style**: Management by Project
- **Structure**: Permanent and temporary organizations are used side by side.
- **Shared Values**: XiTrust employees value customer and team orientation. Most employees are empowered to manage their own work inside projects as temporary organization. Learning of new technologies and methodologies is encouraged although it often has to be neglected due to shortcomings in scheduling. Named values are also reinforced by the XiTrust strategy statement (XiTrust, 2008).
- **Strategy**: XiTrust's foremost strategy statement or vision is to create outstanding customer value (XiTrust, 2008).
- **Skill and Staff**: Staff uses intensive coordination and works self managed.
- **Systems**: Systems are not yet well-defined. They are created and used as needed.

Gareis (2001) names the listed characteristics in Style, Structure and Shared Values as essential for project-oriented organizations.

In a project-oriented organization a great amount of projects is executed over the years. Additionally, each customer has different needs, approaches and business domains. Nevertheless, three repeated patterns or types of customer projects can be observed at XiTrust. The types are sorted by degree of complexity in ascending order.

I: Installation & Configuration. The project delivers a standard installation of a XiTrust product, customized by workflow configuration. Development of an individual command may be included. Project work is service intensive. Project duration is about a month.

For further information about XiTrust product vocabulary refer to Keuschnigg (2007).

II: Installation & Module. Project delivery includes a standard installation of a XiTrust product enhanced with a module not yet available, e. g. a module for a third party software connection. The amount of development work is equal or slightly higher than the amount of service work. Project life time is up to three months.

III: Development. The project requires development of two or more modules up to a whole new software product. Project work is development intensive and projects last four months or more.

Most projects are of type I and II. Additionally internal undertakings are set up as projects. Those range from developing and implementing backup concepts to marketing campaigns. Hence, commonalities hardly exist.

Based on the organizational characteristics and project types the following areas to be improved were identified during the analysis and experiment.

Resource Planning. For projects, resources are scheduled based on the average availability of individual employees. It is not or sparsely accounted for time the employee is scheduled for other projects, routine tasks and administrative matters. Furthermore, no central capacity planning tool exists to collect and look up resource availability data.

Additionally, most projects are executed by only one or two XiTrust employees. Due to the resulting poor knowledge dispersion it is difficult to even out peak demands or cope with unexpected absenteeism and urgent support work.

Project Management. Analyzing past XiTrust projects only few commonalities can be found in project management. Usually a workshop is conducted at the outset of each project which is used to some extend as sales, training and requirements elicitation workshop. The workshop is followed by an implementation, a delivery and a sometimes neglected closure phase. A detailed description of the existing course of a project can be found in Richter (2009).

A project is managed by one of three dedicated project managers. Since no standardized project procedure or framework exists each employee uses different project management practices for planning, monitoring and control based on the individual professional background and education. As a result it is difficult to step in for another project manager and understand and resume his work.

In addition, the XiTrust project managers act as salesmen and ongoing contact person for their customers. They serve a customer from sales to end of project and are the main contact person for all his requests and new project ideas. Thereby, each employee gains a deep understanding of his customers and each project is set up and managed based on special customer needs and project type. On its own, this is beneficial behavior, but paired with the fact a standardized project

procedure or framework does not exist, it adds complexity and hampers cross-project coordination and learning.

Project Portfolio Management. Many problems in project portfolio management are a direct result from the problems named above. Since no resource availability data is available, resource allocation and scheduling for more than one project is difficult and often incorrect. As a result, employees tend to be overloaded and project deadlines slip regularly. Slipped project deadlines then cascade into slips in new projects because unfinished work and rework was not accounted for in resource scheduling.

Due to incompatible project management practices coordination between project managers is weak. Knowledge about customers, project progress and special challenges is not spread, errors are repeated. Cross-project solutions and procedures are not developed.

Furthermore, an overview of important data and schedules for all projects does not exisit. It is not possible to quickly look up how many projects are running or scheduled in parallel for any point in time. Hence, it is hardly possible to determine when best to schedule or not to schedule new projects, which projects may experience resource shortages or prioritize between projects.

Project-Centered Development. For most customers individual modules are developed and rarely integrated back into the main product line, although XiTrust's strategy asks for development of a reusable and resalable product line. Hence, software development is project-centered. As a result the product line itself is stagnant and, due to the need to support many different bits and pieces, support costs soar.

Additional trouble caused the transition from the old product version (e2server) to a newly developed product line (based on e3server). During this transition for some customers the old product version was not suitable anymore while the new version was not yet ready. As a result, even more individual modules were developed.

To improve the depicted situation and remedy named shortcomings, a course of action was agreed upon:

- Develop a standardized project management procedure.
- Develop a standardized software development approach.
- Enable corporate resource and project coordination.
- Reorient software development back to product-centered development.

The specific areas to be improved as well as the exact course of action were not only determined through analysis but also by challenges uncovered during the project experiment as detailed below.

4.2 Project Experiment

The project experiment 'Scrum for ECODES' was, besides the analysis, the first task of the adoption process. ECODES is the short name for a type III project conducted with a major customer. At the outset of the thesis, the project execution phase was about to start and the project situation presented itself as detailed below.

Methodology to be used: Scrum. Since XiTrust had previously experienced troubles with type III projects, a standard methodology was asked for. One employee had heard at a conference Scrum is suitable for such projects. Hence, it was decided to give it a try.

Estimated project duration: 4 months. Although learning about and usage of Scrum was started together with the project, it was limited to project execution, namely use of sprints and Scrum specific workshops. Requirements elicitation and project planning and scheduling was handled with traditional and known methods. As a result, effort estimation and scheduling had not improved.

Developers allocated: 4. Two developers were seasoned developers, two were apprentices. Although the project duration was computed with a staffing of 2.5 developers staffing had been increased already because it was suspected the effort had been underestimated.

Customer: challenging. The customer is one of XiTrust's major customers, but also challenging to conduct projects with. The customer's organization lacked clear responsibility assignments. Therefore up to three people decided on the further course of the project. Additionally, requirements tended to change significantly during the project. Nevertheless, a fixed-priced contract was mandatory.

Risk: high. Besides XiTrust already had experienced problems with projects of that size, ECODES was a high-risk project due to several other risk-factors. Software development tasks included the usage and integration of several technologies new to XiTrust staff. The software product was required to interface with several external components delivered by third parties. The actual enduser of the software was not known or existent at the outset of the project.

Summarized, the initial situation was challenging. The application of Scrum in this context is suitable for several reasons. First, the project was staffed with four developers, one scrum master and one product owner, a headcount inside the recommend range for team size. Second, requirements were vague and had to be elicited and negotiated during ongoing development. Third, the challenging customer constellation required open and high communication and close collaboration. Finally, it is recommended to experiment with agile software development in a complex context, because otherwise the adoption experiment will not reveal all challenges and special requirements which need to be accounted for in the final process layout.

4.2.1 Course of Project

As mentioned, one of the requests for this thesis was to take charge of the ongoing Scrum adoption and fill out the scrum master role. Based on the project situation at hand, the first task was to prepare the team and the project for the first sprints.

For the team, a learning-by-doing approach was chosen. The purpose of each workshop or artifact was explained right before the first use. Furthermore, once a sprint the team had lunch together and discussed different aspects of software development such as testing with mockups or doneness criteria. The 'Scrum Lunch' also helped to bring the team closer together. Additionally, for a couple of sprints 15 minutes every second day were dedicated to learning about a special aspect of Scrum or agile development in general. A training with an external trainer was not held on request of the management.

The first step in preparing the project for sprints was to define requirements as user stories in a step-by-step approach. To begin with, just enough stories for the first two sprints were written. During the sprints the product owner worked on filling the product backlog with all stories in more or less detail. The complete set of user stories was then estimated in an estimation workshop. Afterwards, the project was planned and scheduled anew, using estimation data from the workshop and the average amount of work completed in the first three sprints.

During the further course of the project important events influenced the ongoing adoption of agile software development and the project itself.

New project schedule communicated: Early July. The new project schedule determined a realistic deadline for end of development for Mid-October. In comparison, the old schedule promised a completion for end of August. Following the agile principle of open communication this new deadline had to be communicated to the customer.

This took a lot of pressure off the project, especially the software development part. As a result, it was not only easier to appoint time to the adoption procedure and improvement measures, but also resulted in a software product of higher quality, since too tight schedules tend to result in lower quality DeMarco (2001).

At customer site the schedule adjustment of two months caused quite a stir initially. It was thought about introducing a penalty requlation or even canceling the project. But with keeping communication high and demonstrating workable software at the end of each sprint the situation was calmed down over the next sprints.

The proposal to split the project into two or more releases and deliver a specific part of the software product at end of August was not accepted by customer representatives. Reasons were not given, but it is considered possible the customer's

own concept of the product was not yet clear enough to easily state where to split. Additionally, the customer had not experienced deliveries of working software mid-project and therefore maybe did not consider it possible.

Agreed to continue with Scrum: Mid-August. XiTrust representatives decided to prepare for using agile software development beyond ECODES after already experiencing a series of positive effects in the ECODES project. Basis for this decision was a cross-project retrospective. Three projects including ECODES were scrutinized for problems and strengths in their management. Both, strengths and problems were compared with the capabilities of agile software development exhibited during ECODES. Since most of the problems could be mitigated and strengths retained, the decision was made in favour of agile software development.

End of Development: Mid-October. The end of development was reached in Mid-October, one week after schedule. Nevertheless, customer representatives were highly satisfied with the finished software product. The additional week was easily accepted, since it was caused by negligence by the customer and his other vendors.

End of Project: Early December. Due to the regular review every two weeks the software system was approved at the first acceptance test. Customer approval is followed by billing and payment which is induced earlier, after a total project lifetime of 6 months, than for previous type III projects.

In addition to those significant events, a couple of other factors affected the project. Particularly, the many dependencies on third party software development organizations were an ongoing issue. Most of their deliveries were late or unsuitable. A lot of effort went into producing workarounds.

Internally, other projects repeatedly disrupted sprints. Some finished late. In others problems cropped up and needed immediate attention. While it is possible to schedule a buffer for interruptions beforehand, after this buffer is used up the sprint is affected. Being not able to complete a sprint without being responsible for the causes is demotivating.

Regarding staff, at first knowledge about agile software development had to be build up to increase support and counteract skepticism. Additionally, the team comprised two senior developers and two apprentices. Hence, the skill gap between those two groups was quite significant which made it difficult to distribute the workload evenly between all team members. To slowly reduce the gap senior developers assigned tasks of increasing difficulty to the apprentices and supported them with execution. This contradicts Scrum rules, but is a necessary measure for future development. In fact, an approach to even out skill levels planned in advance would be even more beneficial.

For the Scrum implementation, a decisive factor was a trusted person was put in charge

of the adoption process. The appointee, namely the author, had already worked with the team for about two years and trust in her abilities and knowledge in software development was already built up. Additionally, a person, whose sole responsibility is to keep the adoption process going, precludes letting things slip and superficial implementations. Furthermore, the team is challenged to try practices more than once and continuously improve knowledge and execution.

Essential for the overall success of both, the adoption of agile software development and the project, was not only training in agile software development was deemed important, but also team building and the social aspects was included in the adoption strategy. Special retrospectives and a fortnightly shared lunch were conducted to bring the team members closer together and establish team work agreements. Change project management was utilized for communications about adoption and handling of resistance and skepticism.

4.2.2 Results

The adoption of Agile Software Development brought about very positive results for the project. The development work was finished on time. All customer parties are highly satisfied with the end result. The outstanding project results even saved XiTrust from loosing that customer. The project was completed earlier than previous type III projects. Especially the effort for rework and project closure was highly reduced.

Internally, the team cohesion and collaboration was increased according to a evaluation from the team members. Additionally, it became visible team members were and still are helping each other out increasingly. A better integration and increasing skills of the apprentices was also noticeable. Important decisions are documented and available for later reference.

The adoption of agile software development contributed to this success substantially through

- **better control of project and individual work packages.** Particularly the requirement to complete features a 100% before counting against progress allows a much better progress control. Paired with two-week iterations actual progress is visible to all stakeholders.
- **continuous feedback.** Agile software development stipulates continuous feedback on many different levels such as testing, product owner, code reviews and the customer. A significant improvement in project progress was noticeable once the customer provided such feedback and project collaboration on a regular basis.
- **continuous team and process development.** Continuous team and process de-

velopment and improvement are an integral part of agile software development. Feedback and retrospectives enable to reveal deficiencies in the adoption of practices through the people who actually use the practices.

- **independency on individual's knowledge.** Knowledge is spread more evenly throughout the team and the whole team is responsible for the completion of work assignments. Hence, the dependency on individuals and their special knowledge is highly reduced.

In conclusion, the adoption of agile software development was successful. Not all practices were accepted and valued initially. But once used in a couple of sprints the benefits became visible and the team adopted the practices in question fully.

4.3 Process and Tool Development

Process and tool development was an ongoing activity while working on adoption and writing this thesis. First software development was stabilized using agile methodologies. Second, project management was standardized and project coordination enhanced by establishing a uniform project process.

4.3.1 Agile Methodologies

The first methodology selected was Scrum. As stated, this decision was predetermined at the outset of this thesis. Since Scrum is applicable for the given context as detailed above, the decision to use it was not reverted. Nevertheless, Scrum is a framework and needed to be augmented with software development and, depending on the context, project management and management practices.

A couple of software development practices (continuous integration, testing, refactoring, collective ownership, coding standards) were already in place at XiTrust. Based thereon and the fact XP is well known and discussed, it was selected as basis for software development practices not explicitly part of Scrum. The utilization of existing practices was improved and new practices and concepts were added (minimize code lines, collocated teams) based on the methodology.

It was not chosen and possible to fully implement XP. Some of the practices, namely planning game, small releases and metaphor overlap with Scrum. An on-site customer was not available since the customer did not name who the end users of the product are. The 40-hour week is part of the Austrian work regulations. The attempt to introduce pair programming in a broader context failed. The resistance was very high and in the authors opinion pushing it through would have jeopardized the whole adoption

process. Nevertheless, it is used for joint code reviews, refactoring and tricky code parts on occasions.

The previously named methodologies both cover a specific perspective of a software development organization, namely project management and software development. For most of the practices reasoning and detailed instructions are available from that perspective only. As a result, acceptance and understanding from both management and development was hindered. Lean Software Development bridges this gap. For all used practices of Scrum and XP the educated reader can uncover their cost, effectiveness and efficiency considerations in Lean Software Development. That knowledge was used to improve understanding of the benefits and support for certain concepts such as the consequences of partially completed work or not yet ordered functionality.

4.3.2 Project Management

Goal Directed Project Management was chosen as basis for the project management process. Due to its properties it blends well with Agile Software Development. GDPM emphasizes the importance of people and their collaboration for project success. Its special PSO perspective is based on experiences with IT projects (Andersen et al., 2004, P. 3). Additionally, in planning levels similar to the planning onion are used. *Big Day* deliveries are discouraged. Finally, a GDPM project starts with the definition of purposes and goals which can be compared to formulating a product mission.

For the final process definition, standard project closure activities such as project learning, post project calculation and project closure meeting (Schelle, 2004, P. 257) were slightly altered to fit into agile software development. Additionally, the concept of basic orientation from change project management was used to improve purpose and goal setting for projects.

The project management process was introduced through a project simulation. A new project was selected and all project managers jointly set up and planned the project using the new process step for step. Afterwards, the execution and closure phase were run through theoretically. The results were, all project managers are familiar with and understand the purpose of all steps and elements of the new process. Additionally, they are able to scale the process according to project size.

4.3.3 Product Development and Project Coordination

After the success with Scrum for ECODES it was decided to use it for all projects. But most XiTrust projects are small and were staffed with one or two employees only. Such a small team size is not recommended in agile software development. Additionally, projects running in parallel would disrupt each other's sprints. Hence, the challenge

was how to further use agile software development in this setting without generating too many issues.

In development of Scrum many thoughts went into scaling it for multiple teams. But, information about how to run multiple projects with one team is scarce. Dong et al. (2008) even states little research is available for the whole area of multi-project management in software development. As a consequence, XiTrust's own approach had to be developed.

Most XiTrust projects develop enhancements for their major product the e3server. Therefore, the team now develops the product not individual projects. Requirements for the product are generated from projects, support requests and internal ideas and consolidated into one product backlog. A product owner is assigned to overview the overall product development, called product owner XiTrust.

Nevertheless, the individual projects need to be managed and coordinated. A product owner is assigned for each project. Project product owners have to discuss their requirements with the product owner XiTrust before they are added to the product backlog. Similar requirements are merged into more generic product features which can be sold to more than one customer. The product owner XiTrust also prioritizes projects based on their progress to ensure all projects get their fair share of resources. Internal projects, for instance development of a corporate backup strategy, are treated as customer projects with the CEO as customer.

The project product owner has all responsibilities of a Scrum product owner. Hence, he is also responsible to schedule the project and prioritize the requirements with his customer. Additionally, he has to coordinate his project with other running or scheduled projects. For this purpose, corporate project schedule overview and resource planning were developed. The standardized project process eases coordination because project plans and artifacts are comparable and easily understood by the other product owners.

A few risks are associated with this approach and mitigation approaches had to be found.

- **Coordination between product owners is increased.** The product owner team currently comprises only 2-3 members. Hence, this aspect can be neglected.
- **Developing many different features or projects in one sprint may slow down the team.** This concern is also stated by Nocks (2006). Consequently, the product owner XiTrust needs to keep a sprint homogenous by grouping features by product and developing as little different projects as possible in a specific sprint.
- **Support requests may disrupt sprints.** At XiTrust, the team has to develop bug fixes in addition to other development work. To minimize disruptions from support requests, a careful classification is necessary. Only urgent requests are handled during the sprint, all others are scheduled for upcoming sprints or rejected

if not legitimate.

- **Customers may not be served in time.** With a sprint length of two weeks, the time, customers have to wait for their requests to be completed, averages at three weeks. Currently, this is reasonable for XiTrust customers.

However, the advantages outweigh the possible disadvantages. Workload is distributed evenly between team members and over time. Low workload in one project is balanced with higher workload in other projects. The product itself is constantly enhanced. Newly developed or scheduled features may even be sold to other customers since the product backlog can serve as product roadmap. Support staff only needs to support a single product rather than a range of project specific developments. Bugs at one customer site are fixed in the product. Consequently, the fixes are automatically available for other customers.

4.3.4 Adoption Project

The adoption project was set up to structure and streamline the adoption of the new processes. It was started after the decision to move on with the adoption of agile software development. For the adoption project, the concepts of GDPM were used and thereby introduced to XiTrust staff.

First, a mission breakdown structure was defined. The main purpose everyone agreed on, was to achieve an *effective, controllable software development with motivated employees and satisfied customers*. Detailed purposes reinforced the product orientation of software development and named the areas to work on: project management, software development, support and innovation.

The goal was to have and use a lightweight process for each area. The processes for project management and software development were developed in this thesis as described above. Their description is part of a XiTrust organization handbook. The complete handbook is the final delivery of the adoption project. It is written in German, since this is the main language used in the organization. Furthermore, it serves as basis for further process improvement at XiTrust. The support and innovation processes were not part of this thesis. Nevertheless, suggestions what to include are given. The parts of the handbook developed in this thesis are attached as appendix A.

Chapter 5

Summary and Prospect

To conclude, this chapter summarizes the most important findings. Additionally, suggestions how to enable ongoing improvements are given.

5.1 Summary

The adoption of agile software development is not an easy endeavor. It has to be dealt with skepticism from employees and customers. Some practices seem counterintuitive at first and are rejected. Consequently, the coach or person responsible for the adoption process has to act tactfully but with emphasis. For this thesis approaches from change project management were used to ease the adoption process.

The adoption is further complicated, if the context of the organization is not originally ready for agile software development. At XiTrust, projects and thereby team sizes were too small for orderly teamwork and uninterrupted iterations. Hence, in addition to the adoption of agile software development a part of the organization had to be reorganized. Now one team works at multiple projects concurrently.

The approach 'one team - multiple projects' is scarcely studied and almost no reference material available. Hence, a special approach for XiTrust had to be found. In short, development is now organized around products not around projects. Product development enables bigger team sizes and working on more work packages concurrently. All project requirements are dealt with as product requirements in one single backlog. The products and projects are managed by a team of product owners in tight collaboration.

The developed approach poses major benefits, but it has to be used with care. Increased coordination effort and inhomogeneous sprints may slow down progress or even cancel out benefits. In conclusion, agile software development was successfully

adopted, but XiTrust employees have yet to invest some effort to avoid pitfalls and utilize their special approach best.

The work at XiTrust resulted in closely related process definitions for each software development and project management. The final description of the developed processes and references to further reading is provided in an organization manual. To enable ongoing enhancements, the manual was written in German. Thus, it is attached as appendix A.

5.2 Prospect

Literature about adoption processes often notes a complete change-over to agile software development may last three to five years (Schwaber, 2007; Poppendieck and Poppendieck, 2007). The new practices may reveal technical inadequacies which cannot be resolved immediately. Each new project may bring about new challenges. Even after a complete adoption constant optimization is required because the organization and its context change over the years.

At XiTrust, two areas need further attention to best build on the basis established with this thesis. Both project management and software development need a person responsible for ongoing improvement. In Scrum, the scrum master role includes those responsibilities. Hence, the software development process is catered for, but an appointee for project management is still required.

Additionally, some practices need to be enhanced further to support the established process best. Particularly, testing activities such as unit testing and especially integration testing need further attention. Furthermore, release management of the e3server is currently only vaguely defined.

List of Figures

List of Tables

List of Abbreviations

C3	Chrysler Comprehensive Compensation System; a new payroll system intended to replace all existing payroll systems at Chrysler
DSDM	Dynamic Systems Development Method
ECODES	Name of the project executed during the adoption experiment
FDD	Feature Driven Development
GDPM	Goal Directed Project Management
GmbH	Gesellschaft mit beschränkter Haftung, translated: limited liability partnership
LD	Lean Development
LSD	Lean Software Development
PSO	People, System and Organization
XP	eXtreme Programming

Bibliography

5am Solutions (2008). Guideline: Reaching consensus - 5 finger vote. https://intranet.5amsolutions.com/display/process/Reaching+Consensus+-+5+Finger+Vote.

Agile Alliance (2001). The manifesto of agile software development. http://agilemanifesto.org. Last accessed: 18.11.2009.

Agile42 (2009). The scrum tool agilo. http://www.agile42.com/cms/pages/agilo/. Last accessed: 02.09.2009.

Andersen, E. S., Grude, K. V., and Haug, T. (2004). *Goal Directed Project Management*. Kogan Page, London, Philadelphia, third edition edition. ISBN:9780749441869.

Beck, K. (2000). *eXtreme Programming explained - Embrace change*. Addison-Wesley, Boston et. al. ISBN:201616416.

Beck, K. and Saff, D. (2009). JUnit.org Resources for Test Driven Development. http://junit.org. Last accessed: 22.09.2009.

Boehm, B. (2002). Get Ready for Agile Methods, with Care. *in: Computer*, 35(1):64–69.

Boehm, B. (2006). A view of 20th and 21st century software engineering. In *in: Proceedings of the 28th international conference on Software engineering*, pages 12–29.

Boehm, B. and Turner, R. (2003). *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, volume Volume 2753/2003 of *Lecture Notes in Computer Science*, chapter in: Rebalancing Your Organization's Agility and Discipline, pages 1–8. Springer Berlin, Heidelberg. ISBN: 978-3-540-40662-4.

Boulevart (2009). Why & how to write user stories. http://scrum.boulevart.be/?p=53. Last accessed: 11.09.2009.

Burn, O. (2009). Checkstyle 5.0. http://checkstyle.sourceforge.net/. Last accessed: 23.09.2009.

Buschermöhle, R., Eekhoff, H., and Bernhard, J. (2008). Erfolgs- und Misserfolgsfaktoren bei der Durchführung von Hard- und Softwareentwicklungsprojekten in Deutschland.

Carr, J. (2008a). Evolving retrospectives: Applying themes. `http://blog.james-carr.org/2008/09/26/evolving-retrospectives-applying-themes/)`. Last accessed: 15.06.2009.

Carr, J. (2008b). Retrospective patterns. `http://blog.james-carr.org/2008/09/04/retrospective-patterns/`. Last accessed: 04.09.2008.

Cohn, M. (2006). *Agile Estimating and Planning*. Prentice Hall. ISBN:.

CollabNet. Hudson. `https://hudson.dev.java.net/`.

CollabNet. Subversion. `http://subversion.tigris.org/`.

Cooper, A. and Reimann, R. M. (2003). *About Face 2.0: The Essentials of Interaction Design*. Wiley & Sons. ISBN: 9780764526411.

Davis, A. M. (1993). *Software Requirements Revision - Objects, Functions, & States*. PTR Prentice Hall, Englewood Cliffs, New Jersey. ISBN: 013805763X.

DeMarco, T. (2001). *Spielräume - Projektmanagement jenseits von Burn-out, Stress und Effizienzwahn*. Carl Hanser Verlag München Wien. ISBN: 3446216650.

Derby, E. and Larsen, D. (2006). *Agile Retrospectives - Making Good Teams Great*. The Pragmatic Bookshelf, Raleigh, Dallas.

Dong, F., li, M., Zhao, Y., Li, J., and ye, Y. (2008). *Making Globally Distributed Software Development a Success Story*, chapter in: Software Multi-project Resource Scheduling: A Comparative Analysis, pages 63–75. Springer Berlin / Heidelberg. ISBN: 9783540795872.

Edgewall Software (2009). The trac project. `http://trac.edgewall.org/`. Last accessed: 11.09.2009.

Elssamadisy, A. (2009). *Agile Adoption Patterns - A Roadmap to Organzational Success*. Addison-Wesley, Upper Saddle River et. al. ISBN:9780321514523.

Gareis, R. (2001). Programmmanagement und Projektportfolio-Management: Zentrale Kompetenzen Projektorientierter Unternehmen. *in: Projektmanagment*, 2001/01:4–11.

Goldratt, E. M. (1997). *Chritical Chain*. Gower Publishing Ltd. ISBN: 9780884271536.

Haberfellner, R. (2008). Skriptum Unternehmungsführung und -organisation.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. The Agile Software Development Series. Addison-Wesley, Boston et.al. ISBN:0201760436.

Highsmith, J. and Cockburn, A. (2001a). Agile software development: The business of innovation. *in: Computer*, 34(9):120–122.

Highsmith, J. and Cockburn, A. (2001b). Agile software development: The people factor. *Computer*, 34(11):131–133.

Keuschnigg, E. (2007). Optimierung des Software Releaseprozesses unter Anwendung von automatisch generierten Testdaten.

Kniberg, H. (2007). *Scrum and XP from the Trenches*. C4Media Inc. ISBN: 9781430322641.

Larman, C. and Basili, V. R. (2003). Iterative and incremental development: A brief history. *in: Computer*, 36(6):47–56.

Lüsch, F. and Zitzke, E. (2004). *Projektleitung - Alle Rollen souverän meistern*. Carl Hanser Verlag München Wien. ISBN: 2446228233.

Martin, R. C. (2008). *Clean Code - A Handbook of Agile Software Craftmanship*. Robert C. Martin Series. Prentice Hall, Upper Saddle River, NJ et. al. ISBN: 9780132350884.

Mountain Goat Software (2005). Learning scrum. `http://www.mountaingoatsoftware.com/scrum-figures`. last accessed: 21th August 2009.

Murgridge, R. and Cunningham, W. (2009). The fully integrated standalone wiki, and acceptance testing framework. `http://fitnesse.org`. Last accessed: 22.09.2009.

Nocks, J. (2006). Multiple simultaneous projects with one extreme programming team. In *in: Proceedings of the conference on AGILE 2006*, pages 170–174. ISBN: 0769525628.

Pichler, R. (2008). *Scrum - Agiles Projektmanagement erfolgreich einsetzen*. dpunkt.verlag, Heidelberg. ISBN: 9783898644785.

Poppendieck, M. and Poppendieck, T. (2007). *Implementing Lean Software Development - From Concept to Cash*. The Addison Wesley Signature Series. Addison-Wesley, Upper Saddle River et. al. ISBN:0321437381.

Richter, E. (2009). Structured Requirements Specification and Automatic Validation Testing for Workflows.

Robbins, S. P. and Barnwell, N. (2002). *Organisation Theory - Concepts and cases*. Prentice Pearson Education Australia, Frenchs Forest, 4th ed. edition. ISBN: 1740095456.

Schelle, H. (2004). *Projekte zum Erfolg führen, 4. Auflage*. dtv, 4 edition. ISBN: 3423058889.

Schwaber, K. (2004). *Agile Project Management with Scrum*. Microsoft Press. ISBN: 9780735619937.

Schwaber, K. (2007). *Scrum im Unternehmen*. Microsoft Press Deutschland. ISBN: 9783866456433.

Shore, J. (2008). The decline and fall of agile. `http://jamesshore.com/Blog/The-Decline-and-Fall-of-Agile.html`.

Spolsky, J. (2009). Fruity treats, customization, and supersonics: Fogbugz 7 is here. http://www.joelonsoftware.com/items/2009/07/20.html.

Sutherland, J. (2006). Why the three questions in the daily scrum meeting? http://jeffsutherland.com/scrum/2006/06/why-three-questions-in-daily-scrum.html. Last accessed: 02.09.2009.

Womack, J., Jones, D. T., and Roos, D. (2007). *The Machine That Changed the World*. Free Press, New York, reprint edition. ISBN: 9780743299794.

XiTrust (2008). Xitrust strategy statement.

Yip, J. (2006). It's not just standing up: Patterns of daily stand-up meetings. http://www.martinfowler.com/articles/itsNotJustStandingUp.html. Last accessed: 02.09.2009.

Appendix A

Organisationshandbuch XiTrust

Das Organisationshandbuch wurde zur Verwendung und Weiterentwicklung im Unternehmen entwickelt. Das hier präsentierte Format ist eine Abschrift des im Unternehmen verwendeten, digitalen Formats. Der Text des Anhangs folgt daher nicht den Formalkriterien des Hauptteiles der Arbeit.

A.1 Einleitung

Dieses Dokument enthält die Prozessdefinition für XiTrust nach Stand 2009. Es ist die Basis für die unternehmensweite Einführung agiler Entwicklungs- und Organisationsprinzipien sowie entsprechenden Praktiken. Daher ist es in Deutsch verfasst. Die abgedeckten Bereiche reichen von der allgemeinen Organisation, über die Softwareentwicklung und Projektabwicklung bis zu Support und Innovation.

Das Wort Basis weist darauf hin, dass dieses Dokument als Grundlage für eine kontinuierliche Weiterentwicklung des Unternehmens und seiner Prozesse und deren Beschreibung dient. Die Abschnitte A.3.6, A.3.6, A.4.5 und A.6 beschreiben, wie sowohl das Produkt, als auch der Prozess kontinuierlich verbessert werden können. Dies entspricht dem jeder agilen Methode zugrundeliegenden Prinzip anpassungsfähig zu sein (Highsmith, 2002, S. 199ff).

A.1.1 Change Agents

Es ist zu empfehlen, einem Mitarbeiter oder Mitarbeiterteam die Rolle eines Change Agents zuzuweisen und ihm auch Zeit für die Ausübung dieser Rolle zuzugestehen. Der Change Agent achtet darauf, dass die bereits erlernten neuen Verhaltensweisen aufrecht bleiben. Er kämpft damit gegen den in (Schwaber, 2007, S. 25-32) beschriebenen *Muskel-Memory-Effekt*, der für das Zurückfallen in alte Verhaltensweisen verantwortlich ist.

Der Change Agent ist auch für das Ausprobieren und Etablieren neuer Verhaltens-

weisen verantwortlich. Gegen manche neue Methoden wird innerlicher Widerstand bestehen, da sie scheinbar vorhandener Logik und Verständnis widersprechen. Beispiele hierfür sind das "doppelte" (digitale und analoge) Führen des Sprint Backlogs oder Pair Programming. Manchmal wird es für die Weiterentwicklung der Organisation auch notwendig sein, sich von Liebgewonnenem zu trennen.

Im Entwicklungsbereich ist die Rolle des Change Agents beim Scrum Master angesiedelt. Dieser ist laut seiner Rollenbeschreibung bereits für die Erfüllung dieser Aufgaben verantwortlich. Für das Management könnte der Product Owner XiTrust (siehe A.3.4) mit der entsprechenden Autorisierung oder der Geschäftsführer selbst diese Rolle übernehmen.

A.1.2 Organisationsweites Lernen

Weiters ist es wichtig einen stetigen Lernprozess zu etablieren. Nur mit dem notwendigen Wissen über die agilen Prinzipien und Praktiken ist es möglich, diese vollständig umzusetzen und in weiterer Folge situationsgerecht anzupassen. (Elssamadisy, 2009, S. 5-12) geht sogar so weit zu behaupten, das Lernen sei der Engpass einer jeden agilen Softwareentwicklungsanstrengung. Dies unterstreicht, wie wichtig es ist, sein Wissen zu erweitern.

Lernen findet in einem Unternehmen auf verschiedene Weise statt, einerseits als Erweiterung des theoretischen Wissens und andererseits bei der praktischen Umsetzung dieses Wissens. Im zweiten Fall ist es notwendig, das dabei generierte Wissen für spätere Referenz zu konservieren, um Lernprozesse nicht teilweise oder vollständig wiederholen zu müssen.

Abschließend ist zu bemerken, dass die vollständige Umstellung des Unternehmens sich über einen Zeitraum von drei bis fünf Jahren erstrecken kann. Dies ist begründet durch den obengenannten Muskel-Memory-Effekt, aber auch der Tatsache, dass durch das Lösen eines Problems andere zu Tage treten werden.

A.2 Allgemeine Organisation

Die allgemeine Organisation definiert Prozesse für die Themen Abwesenheit, Zeiterfassung, Office-Abdeckung und Kundenanfragen. Diese Prozesse wurden als Teil des Transformationsprojektes (siehe 4.3.4) behandelt, aber nicht vom Autor dieser Arbeit definiert.

A.3 Softwareentwicklung

A.3.1 Einleitung

Dieser Abschnitt beschreibt die wichtigsten Abläufe und Aufgaben in der Softwareentwicklung von XiTrust. Er dient als Schnellreferenz, daher sind alle Beschreibungen absichtlich kurz gehalten. Weiterführende Informationen können der angegebenen Literatur und den Anhängen entnommen werden.

Voraussetzung ist Basiswissen in der iterativen Softwareentwicklung, Scrum (siehe (Pichler, 2008)), grundlegenden Softwareentwicklungs-Praktiken und Werkzeuge. Weiters wird besonders den Personen, die genannte Rollen innehaben, aber auch anderen Interessierten folgende Literatur empfohlen:

- Team: Agile Software Development Ecosystems (Highsmith, 2002)
- Management: Implementing Lean Software Development (Poppendieck and Poppendieck, 2007)
- Product Owner XiTrust und Product Owner Kunde: Agile Estimating and Planning (Cohn, 2006)

Alle bei den einzelnen Themen angegebenen Referenzen sind nach ihrer Empfehlung geordnet.

Die folgenden Abschnitte sind als zwei große Themenblöcke zu verstehen. Abschnitte A.3.2 - A.3.3 beschreiben hauptsächlich theoretische, für die Anwendung des Entwicklungsprozesses wichtige Konzepte und Begriffe. Abschnitte A.3.4 - A.3.10 beschreiben XiTrust-spezifische Abläufe und Konzepte.

A.3.2 Begriffe und Konzepte

Im Folgenden sind die Begriffe und Konzepte aus Scrum beschrieben, die, im Vergleich zum bisherigen Entwicklungsprozess, neu sind.

Product Owner (PO)

Der Product Owner ist der Repräsentant des Kunden im Projekt. Er betrachtet das Projekt aus Kundensicht. Seine Aufgaben sind:

- Berücksichtigen und Vertreten der Interessen der Stakeholder
- Erstellen, Pflegen und Kommunizieren der Produkt Vision
- Erstellen und Pflegen des Product Backlog
- Maximieren des ROI des Kunden

- Erstellen und Pflegen des Releaseplans
- Enge Kommunikation mit Team und Kunde

REFERENZEN: (Pichler, 2008, S. 9-13)

Scrum Master (SM)

Der Scrum Master ist Coach, Mentor, Moderator und verantwortlich für die Einhaltung und Weiterentwicklung des Prozesses. Seine Aufgaben sind:

- Moderieren und Leiten der Scrum Meetings
- Unterstützen der Kommunikation zwischen Team und Product Owner
- Festigen und Unterstützen der Anwendung der Scrum Prinzipien
- Abschirmen des Teams von externen Aufgabenstellungen und Änderungen des Sprint Backlogs
- Beseitigen von Blockaden und Hindernissen
- Definieren von und Berichten über die Team Produktivität

Die Scrum Master und Product Owner Rolle darf nicht von ein und derselben Person ausgeübt werden.

REFERENZEN: (Pichler, 2008, S. 19-23)

Team

Das Team erstellt jedes Produktinkrement basierend auf Produktvision, Produkt-Backlog und Sprintbacklog. Ein Scrum Team ist immer aus Personen aus mehreren Fachgebieten zusammengesetzt. Es gibt keine absoluten Spezialisten. Jeder kann prinzipiell jede Aufgabe erledigen. Teammitglieder können nur an den Sprintgrenzen ausgetauscht werden. Die Aufgaben des Teams sind:

- Selbstbestimmen des Commitment für einen Sprint
- Selbstständiges Einteilen der durchzuführenden Arbeiten
- Schätzen des Aufwands für die Arbeitsinhalte
- Verwalten und Abarbeiten des Sprint Backlogs
- Entwickeln und Erstellen des Produktinkrements
- Selbstständige Qualitätskontrolle
- Auswahl und Anwenden der notwendigen Entwicklungspraktiken (Testen, Reviews, etc.)

REFERENZEN: (Pichler, 2008, S. 13-19)

Produktmission

Die Produktmission (Pichler, 2008, S. 30-31) dient als Leitbild für die gesamte Entwicklungsdauer eines Releases oder Projekts. Sie beschreibt das Kundenbedürfnis, die wesentlichen Leistungsmerkmale (z.B. Usability, Performance oder Ausfallssicherheit) der Software, sowie eventuell die wichtigsten Funktionen. Sie wird vom Kunden mit Hilfe des PO formuliert.

Die Produktmission hilft abzuwiegen, auf welche Softwaremerkmale (z.B. Performance) in der Entwicklung besonders wert gelegt wird und wo eventuell im Sinne der ROI-Maximierung zurückgesteckt werden kann. Außerdem kann sie als Entscheidungshilfe beim Priorisieren dienen und damit eine Funktion gegenüber einer anderen zurückstellen oder ganz aus dem Projektumfang verdrängen.

BEISPIELE: (Spolsky, 2009) (Pichler, 2008, S. 31)

Produkt-Backlog

Das Produkt-Backlog enthält alle Anforderungen für die Softwareentwicklung. Das Werkzeug für die Verwaltung des Produkt-Backlogs bei XiTrust ist das Ticket System Trac (Edgewall Software, 2009), erweitert mit dem Scrum-Plugin Agilo (Agile42, 2009). Das Produkt-Backlog wird vom Product Owner verwaltet.

Die Anforderungen werden in Form von User Stories beschrieben. Je nach Entwicklungsfortschritt existieren die User Stories in unterschiedlicher Granularität. Zu Beginn eines Projekts oder Releases befinden sich viele Stories noch im Rohzustand und beschreiben einen größeren Arbeitsumfang.

Bevor eine Story ins Sprint-Backlog aufgenommen werden kann, muss sie so aufbereitet sein, dass sie in wenigen Arbeitstagen fertigstellbar ist. Außerdem muss genau feststellbar sein, wann eine Story fertig ist. Dazu werden einerseits, die in einer Story zu definierenden Akzeptanzkriterien und anderseits die Checkliste für Fertigstellung (siehe A.3.7) von Stories herangezogen. Stories können außerdem noch thematisch gruppiert werden.

Jede Story im Product Backlog hat eine eindeutige Priorität, dargestellt als frei wählbare Zahl, beginnend bei 1 als höchste Priorität. Die Priorität einer Story wird bestimmt vom Wert der Story für den Kunden, den Kosten für die Entwicklung der Story und dem Risiko bei der Entwicklung. Methoden zum Festlegen der Priorität werden in (Pichler, 2008, S. 38-43) und (Cohn, 2006, S. 79-120) vorgestellt.

Abbildung A.2 zeigt die Darstellung aller genannten Elemente in Trac. Punkt A.4.2 beschreibt die graduelle Erstellung eines Produkt-Backlogs. Diese Vorgangsweise kann sowohl für Projekte, einzelne Releases oder neue Produkte verwendet werden.

Die im Product Backlog vorgehaltene Arbeitsmenge sollte die in 3-5 Monaten bewältigbare Arbeitsmenge nicht überschreiten. Ist die vorgehaltene Arbeitsmenge zu groß, ist die Wahrscheinlichkeit hoch, dass Stories veralten und überarbeitet oder ent-

fernt werden müssen. Zusätzlich sinkt die Kundenzufriedenheit, wenn sie zulange auf angeforderten Änderungen warten müssen.

REFERENZEN:

- Produkt-Backlog: (Pichler, 2008, S. 34-36) (Elssamadisy, 2009, S. 81-86)
- User Stories: (Pichler, 2008, S. 44-48) (Boulevart, 2009) (Cohn, 2006)

Sprintziel

Das Sprint Ziel dient als Leitbild für einen Sprint, ähnlich wie die Produktmission für ein Projekt oder Release. Es wird vom PO für den zu planenden Sprint formuliert und mit dem Kunden und dem Team abgestimmt.

Wenn während eines laufenden Sprints klar wird, dass nicht alle geplanten Stories umgesetzt werden können, kann anhand des Sprintziels abgewogen werden, welche Story aus dem Commitment herausgenommen wird.

REFERENZEN: (Pichler, 2008, S. 88)

Commitment

Das Commitment ist die Menge der Stories zu deren Umsetzung in einem Sprint das Team sich verpflichtet. Das Commitment wird immer vom Team bestimmt und darf auf keinen Fall von PO oder SM vorgegeben werden. Ist sich das Team nicht sicher ob das Commitment erreicht werden kann, werden solange Stories entfernt, bis sich das Team zu 100% verpflichten kann. Das Team wählt nur jene Arbeitsmenge, für die es sicher ist, dass sie auch im vorgegebenen Zeitraum fertiggestellt werden kann. Damit wird verhindert, dass die Arbeitsmenge die vorhandene Kapazität überschreitet.

REFERENZEN: (Pichler, 2008, S. 99)

Sprint-Backlog

Das Sprint Backlog (Pichler, 2008, S. 102-106) enthält ebenfalls Stories, zusätzlich jedoch auch die für die Umsetzung in ein Produktinkrement notwendigen Tasks. Das Werkzeug für die Verwaltung des Sprint-Backlogs bei XiTrust ist ebenfalls Trac mit Agilo (Abbildung A.3). Zusätzlich wird für die bessere Visualisierung (Elssamadisy, 2009, S. 157-160) ein Whiteboard (Abbildung A.4) verwendet. Das Sprint-Backlog wird vom Team verwaltet.

Auch das Sprint-Backlog ist priorisiert. Die Priorität der Stories wird vom Product Owner vorgegeben, das Team berücksichtigt die Abhängigkeiten der Tasks untereinander. Gibt es Abhängigkeiten zwischen den Stories, muss eventuell die Priorität der Stories in Absprache mit dem Product Owner nochmals geändert werden. Die am Sprintboard

so priorisierten Stories und Tasks werden dann zuerst von links nach rechts, dann von oben nach unten abgearbeitet (Top-Load-Prinzip).

Burndown Charts

Burndown Charts dienen der Fortschrittskontrolle. Der Begriff Burndown ist in diesem Zusammenhang mit Fortschritt gleichzusetzen. Die Charts werden zu bestimmten Zeitpunkten aktualisiert, der tatsächliche Fortschritt mit dem antizipierten Fortschritt verglichen und, je nach Ergebnis, Maßnahmen ergriffen. Üblicherweise weist ein Burndown Chart fünf Eigenschaften auf:

- Die X-Achse entspricht dem überwachten Zeitraum
- Die Y-Achse entspricht der zu erledigenden Arbeitsmenge
- Eine Linie für den idealen Burndown
- Eine Linie für den tatsächlichen Burndown
- Eine Trendlinie für den tatsächlichen Burndown

Der Schneidepunkt der X-Achse und der Trendlinie zeigt an, zu welchem Zeitpunkt zu erwarten ist, dass die geplante Arbeitsmenge fertiggestellt ist. Meist werden zwei verschiedene Burndown Charts verwendet.

Das **Sprint Burndown Chart** (Pichler, 2008, S. 117-118) dient der Fortschrittskontrolle während des Sprints. Die X-Achse repräsentiert die Dauer eines Sprints, die Y-Achse die Summe der geschätzten Stunden für das Commitment. Aktualisiert wird täglich oder öfter. Der im Sprint Burndown Chart ersichtliche Fortschritt wird im Daily Scrum kontrolliert. Wird ersichtlich, dass das Commitment nicht erreichbar ist (die Trendlinie schneidet die X-Achse nach Ende des Sprints), werden Stories mit geringer Priorität aus dem Commitment entfernt. Ist der Fortschritt schneller als erwartet, können neue Stories dem Commitment hinzugefügt werden. Beides muss in Absprache mit dem PO erfolgen.

Abbildung 3.2 zeigt ein Sprint Burndown Chart mit idealem Burndown, tatsächlichem Burndown, der verfügbaren Kapazität und der Trendlinie.

Das **Release Burndown Chart** (Pichler, 2008, S. 69-74) zeigt, wie weit die Fertigstellung eines Releases schon fortgeschritten ist. Es wird am Ende eines jeden Sprints aktualisiert. Die Arbeitsmenge im Release Burndown Chart wird in Story Points angegeben, der Releasezeitraum als eine Anzahl von Sprints. Zeigt die Trendline an, dass das Release nicht im geplanten Zeitraum fertigstellbar ist, kann der PO zusammen mit dem Kunden Stories aus dem Release entfernen.

Manchmal wird das Release Burndown Chart mit einem Release Burnup und einem Chart für die Änderungen in der Arbeitsmenge kombiniert. Ersteres zeigt die Menge der bereits implementierten Story Points und zeigt dem Team den Gesamtfortschritt

an (Poppendieck and Poppendieck, 2007, S. 140). Ändern sich während des Releasezeitraums wesentlichen Teile der Requirements kommt es zu unerwarteten Anstiegen oder Abfällen im tatsächlichen Burndown. Um diese Änderungen auf den ersten Blick sichtbar zu machen, können die Änderungen der Arbeitsmenge als eigene Linie im Release Burndown Chart dargestellt werden.

Die für die Erstellung und Pflege eines Release Burndown Charts verwendeten Werkzeuge können problemlos auf ein Projekt oder Produkt Burndown Chart übertragen werden. Abbildung 3.3 zeigt ein Release Burndown Chart mit idealem, tatsächlichem und antizipiertem Burndown, sowie Burnup und Änderungen der Arbeitsmenge.

Produktinkrement

Das Produkt Inkrement (Pichler, 2008, S. 83-85) (Schwaber, 2007, S. 129) ist das Arbeitsergebnis eines jeden Sprints. Es ist die teilweise Entwicklung des geplanten Softwareproduktes inklusive der bestellten Dokumentation, muss aber immer voll funktionsfähig und auslieferbar sein. Funktionsfähig bedeutet vollständig getestet, d.h. es existieren Tests auf allen Testebenen (Unit-Tests, Integrationstest, Blackbox-Tests, etc.), fehlerfrei und es existiert ein ausführbares Programm. In einem auslieferbaren Inkrement ist jede Funktion durchgängig, d.h. durch alle Ebenen der Software, implementiert. Jedes Inkrement ist außerdem eine Erweiterung des vorangegangenen Inkrements.

Abbildung 3.4 zeigt genannte Prinzipien. Horizontal sind die Ebenen einer Software dargestellt, vertikal die Inkremente. Jeglicher Code, der nicht Teil eines Produktinkrements ist, gilt als partielles Arbeitsergebnis. Partielle Arbeitsergebnisse ähneln dem Inventar einer Produktion. Sie binden Ressourcen, müssen verwaltet werden und können veralten oder verloren gehen. Die Menge der partiellen Arbeitsergebnisse muss daher möglichst gering gehalten werden (Poppendieck and Poppendieck, 2007, S. 24).

Teamkapazität

Die Teamkapazität ist jene Anzahl der Stunden, die das Team in einem Sprint leisten kann. Sie entspricht der Summe der verfügbaren Arbeitsstunden jedes Teammitglieds. Die Arbeitsstunden jedes Teammitglieds errechnen sich wie folgt:

REFERENZEN: (Pichler, 2008, S. 90-91)

Fokusfaktor

Durch den Fokusfaktor werden Schätzunsicherheiten, Störungen und andere Nebentätigkeiten berücksichtigt. Die verfügbare Teamkapazität wird um den Fokusfaktor entsprechend verringert. Der XiTrust Fokusfaktor liegt derzeit bei etwas unter 40%.

	Beispiel
Arbeitsstunden pro Tag	8h
x Anzahl der Tage in einem Sprint	10d
- Abwesenheitszeiten (Urlaub, Zeitausgleich, externe Termine, etc.)	16h
- bekannte andere Verpflichtungen (entwicklungsfremde Aufgaben)	2h
= Arbeitsstunden eines Teammitglieds	**62h**

Tabelle A.1: Berechnung der Arbeitsstunden eines Teammitglieds

Dieser Wert stammt aus den Erfahrungen des Projekt-Experiments ECODES.

Der Fokusfaktor muss von Zeit zu Zeit überprüft und korrigiert werden. Schafft das Team regelmäßig mehr oder weniger als die geplante Arbeitsmenge, wird der Fokusfaktor in 5%-Schritten nach oben oder nach unten korrigiert.

Teamgeschwindigkeit

Die Teamgeschwindigkeit ist ein Maß für die Arbeitsmenge, die ein Team in einem Sprint abarbeiten kann. Sie wird beeinflusst von der Teamkapazität und der Leistungsfähigkeit (siehe Abbildung A.1) des Teams. Die Teamgeschwindigkeit wird in Story Points angegeben und wird für die Errechnung von Releaseplänen und das Beschränken der Arbeitsmenge in der Sprint Planung verwendet.

REFERENZEN: (Pichler, 2008, S. 63-66)

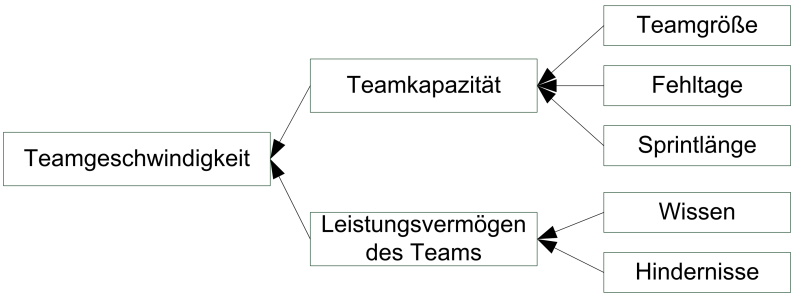

Abbildung A.1: Einflussgrößen der Teamgeschwindigkeit, (Pichler, 2008, S. 64)

Abbildung A.2: Darstellung des Product Backlog in Trac

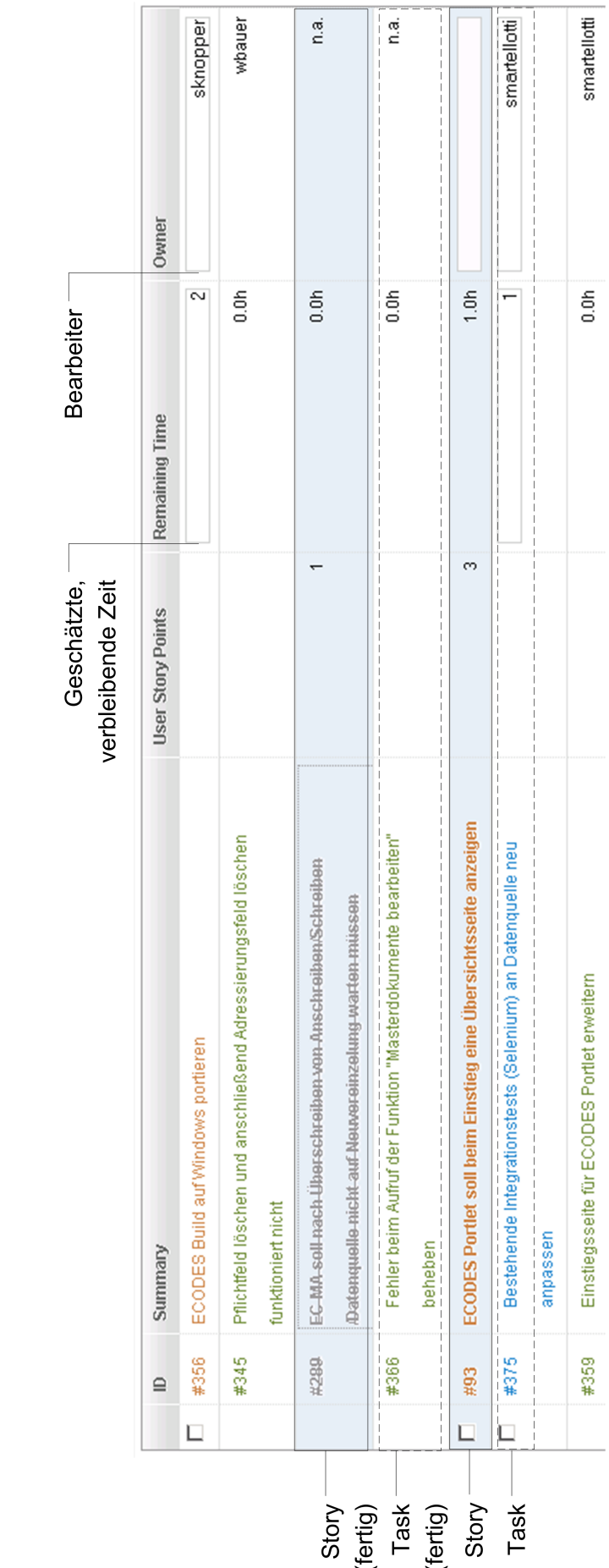

Abbildung A.3: Darstellung des Sprint Backlog in Trac

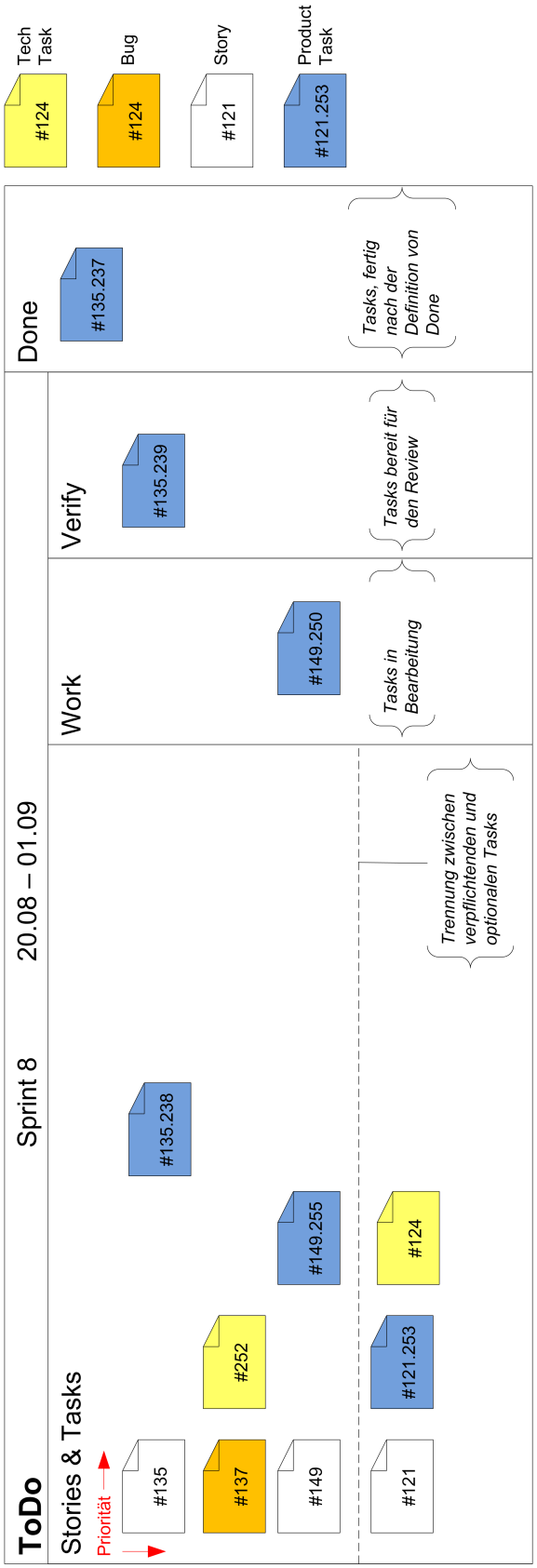

Abbildung A.4: Schematische Darstellung des Sprintboards

A.3.3 Wichtige Stakeholder

Kunde

Der Kunde ist ein wichtiger Stakeholder in jedem Entwicklungsvorhaben. Die Zusammenarbeit mit ihm sollte möglichst auf gegenseitiges Vertrauen gestützt sein. In einer agilen Entwicklungsumgebung ist es außerdem notwendig, dass der Kunde bestimmte Rechte und Pflichten wahrnimmt. Es ist die Aufgabe des jeweiligen Product Owners den Kunden dabei bestmöglich zu unterstützen. Die Rechte und Pflichten des Kunden sind:

- Festlegen des Budgets
- Beschreiben der Produktmission
- Bestimmen über die Priorität der Anforderungen
- Festlegen der Projektinhalte und Nicht-Inhalte (Scope)
- Festlegen der Mindestanforderungen für ein Release
- Festlegen weiterer Anforderungen
- Festlegen des Releasedatums
- Regelmäßige Zusammenarbeit mit seinem Produkt Owner und dem Team zur Klärung von Anforderungen
- Aktives Teilnehmen an den Sprint-Reviews
- Beenden des Projekts, wenn seine Erwartungen vorzeitig erfüllt wurden

Aus diesen Rechten und Pflichten ist ersichtlich, dass gegenseitiges Vertrauen und Ehrlichkeit unabdingbar sind. Andernfalls kann es zu einander widersprechenden Festlegungen kommen, insbesondere im Projektmanagement-Dreieck Scope, Budget und Zeit.

Endbenutzer

Der Endbenutzer als wichtiger Stakeholder wird oft übersehen oder vernachlässigt. Exzellente Ergebnisse bei Kundenzufriedenheit und Kosten-Nutzen-Maximierung sind nur möglich, wenn der Endbenutzer und seine Eigenschaften bekannt sind. Beispielsweise muss die Auswahl von Technologie, Oberflächengestaltung, Benutzerführung und anderen benutzerrelevanten Aspekten der Softwareentwicklung von der Erfahrung der Anwender im Umgang mit Computern beeinflusst werden. Weitere bestimmende Merkmale eines Endbenutzers sind unter anderem

- seine Arbeitsumgebung (z.B. Lärm, Licht, Büro, Arbeitskleidung, verfügbare Eingabegeräte),

- seine Hauptansprüche an die Software (z.B. Geschwindigkeit, Einfachheit, Flexibilität) und
- seine Ziele. Diese können von den Zielen des Kunden wesentlich abweichen.

Zusammen mit dem Kunden sind möglichst früh Repräsentanten der Endbenutzergruppen auszuwählen. Ihnen werden die jeweiligen Produktinkremente zur Verfügung gestellt, ihr Feedback analysiert und in die Weiterentwicklung übernommen. Zu beachten ist, dass bei einem Softwareentwicklungsvorhaben mehrere unterschiedliche Endbenutzergruppen (z.B. System-Administratoren, Daten-Administratoren, Anwender) auftreten können, aus denen dann jeweils ein Repräsentant auszuwählen ist. Strategien zur erfolgreichen Analyse und Zusammenarbeit mit Endbenutzern finden sich vor allem im Fachbereich des Usability Engineerings (Cooper and Reimann, 2003).

Drittfirmen

Drittfirmen, wie Lieferanten, Subunternehmer oder Mitlieferanten, sind ein weiterer wichtiger Stakeholder der Softwareentwicklung. Inadäquate Zusammenarbeit kann zu wesentlichen Problemen in der Projektabwicklung führen. (Buschermöhle et al., 2008, S. 255) listet 'Probleme mit Lieferanten' sogar an erster Stelle der negativen Einflussfaktoren auf Projekte. Insbesondere beim Planen eines Projektes sind die Abhängigkeiten von Drittfirmen zu beachten und einzuplanen. Strategien zur erfolgreichen Zusammenarbeit mit Drittfirmen sind im Critical Chain Project Management (Goldratt, 1997) beschrieben.

A.3.4 Rollen

Product Owner Kunde (POK)

Der Product Owner Kunde ist verantwortlich für die ordnungsgemäße, termintreue und kosten-nutzen-maximierte Abwicklung von Softwareentwicklungsvorhaben mit externen Kunden. Seine Aufgaben entsprechen den des in Scrum definierten Product Owners (siehe A.3.2) und sind:

- Aufbauen der Projektbasis (siehe A.4.2)
- Überwachen und Steuern des Projektfortschritts (siehe A.4.4)
- Abschließen eines Projekts (siehe A.4.5)
- Unterstützen der Kommunikation zwischen dem Kunden und dem Team
- Unterstützen des Kunden bei der Anwendung der inkrementellen Entwicklung
- Unterstützen des Kunden bei der Auswahl von geeigneten Endbenutzer-Repräsentanten

- Planen, Überwachen und Fördern der Zusammenarbeit mit Drittfirmen

Weiters hat er folgende Aufgaben in Absprache mit dem Kunden wahrzunehmen:

- Aufnehmen und Verhandeln des Scopes
- Einteilen der Anforderungen in Releases
- Verhandeln des Releasezeitpunktes
- Verhandeln von Anforderungsänderungen
- Priorisieren jeder Anforderung
- Detailieren der Anforderungen für jeden Sprint
- Planen und Überwachen der Kosten-Nutzen-Rechnung

Product Owner XiTrust (POX)

Der Produkt Owner XiTrust ist verantwortlich für die ordnungsgemäße, termingetreue und kosten-nutzen-maximierte Abwicklung der gesamten Softwareentwicklungsvorhaben von XiTrust. Er repräsentiert den internen Kunden XiTrust. Daher hat er folgende zusätzliche Aufgaben zu denen des POK:

- Überwachen und Steuern des gesamten Entwicklungsfortschritts

Weiters hat er folgenden Aufgaben in Absprache mit den POK und dem internen Kunden wahrzunehmen:

- Sicherstellen, dass alle Softwareentwicklungsvorhaben, auch Kundenprojekte, zur Weiterentwicklung der XiTrust-Produktlinie beitragen
- Priorisieren und Abstimmen der Kundenprojekte mit dem internen Entwicklungsauftrag
- Planen der Ressourcen für die einzelnen Entwicklungsaufträge
- Detailieren der Anforderungen für jeden Sprint

Scrum Master

Die Scrum Master Rolle in der XiTrust Entwicklung entspricht der Scrum Master Rolle in der Scrum Definition (siehe A.3.2).

Team

Das Team ist wie ein Scrum Team zusammengestellt (siehe A.3.2). Die speziellen Verantwortlichkeiten des Teams sind:

- Abwägen von technischen Alternativen in Bezug auf Kosten (als Zeitschätzung) und Konsequenzen (z.B. schnellere Entwicklung vs. einfachere Wartung durch den Kunden)
- Erstellen und Dokumentieren des notwendigen Maßes an Softwaredesign und -architektur zum richtigen Zeitpunkt
- Beschränken des Codes auf die einfachste und genau jetzt benötigte Lösung
- Erkennen von Ursachen und Entwickeln von Lösungen für Probleme im Entwicklungsablauf (z.B. bei hoher Defektrate, bei oftmaligem Fehlschlagen des Builds, Wiederholtes Verfehlen des Sprintziels in hohem Ausmaß, etc.)
- Einhalten und Verbessern der Prozesse
- Auswahl geeigneter Entwicklungspraktiken und -werkzeuge
- Auswahl geeigneter Entwicklungskennzahlen (Teamgeschwindigkeit, Durchlaufzeit von Softwarekorrekturen, etc.) mit Hilfe von SM und POX
- Erstellen notwendiger Dokumentation
- Testen
- Selbstständiges Organisieren der Arbeit zum Erreichen des Sprintziels. Dies beinhaltet insbesondere:
  - Priorisieren der Tasks nach Story Priorität, Risiko und Abhängigkeit
  - Aufteilen der anfallenden Tasks unter den Teammitgliedern
  - Beachten und Beheben von Wissensdifferenzen
  - Pflegen des digitalen und analogen Sprintboards
- Übernehmen von erfolgreichen Experimenten aus der Retrospektive (siehe A.3.6) in die Standards

Kunde XiTrust

Der Kunde XiTrust ist der Kunde der gesamten Softwareentwicklung bei XiTrust. Er ist ein XiTrust Mitarbeiter, aber steuert die Entwicklung wie ein externer Kunde. Seine Aufgaben sind:

- Erstellen einer Produktmission
- Ableiten einer Releasemission für jedes Release aus der Produktmission. Dies ist notwendig, wenn das Projekt über einen längeren Zeitraum geplant ist.
- Beschreiben von neuen innovativen, nicht bereits von Kunden explizit bestellten Anforderungen
- Enge Zusammenarbeit mit dem POX
- Studium des Marktes
- Leiten der Innovationsanstrengungen

Rollenzuordnung

Tabelle A.2 gibt die Zuordnung der Rollen zu den Mitarbeitern an. Es ist ersichtlich, dass zwei Doppelbesetzungen notwendig sind. Diese sollten mit dem Wachstum des Unternehmens eliminiert werden, da die Interessen zweier unterschiedlicher Rollen oft nicht miteinander vereinbar sind.

	Kunde XiTrust	**POX**	**POK**	**SM**	**Team**
Georg Lindsberger	•				
Gregor Karlinger		•	•		
Harald Krassnigg			•		
Wolfgang Bauer				•	•
Gerhard Fliess					•
Markus Wiederkehr					•
Silvio Martellotti					•

Tabelle A.2: Zuordnung der Rollen zu Mitarbeitern

Das für das gesamte Unternehmen abgeleitete Organigramm ist im vollständigen Prozesshandbuch im Abschnitt 'Allgemeine Organisation' zu finden.

A.3.5 Überblick des Ablaufs

Die gesamte zu bewältigende Arbeitsmenge wird in kleinen Schritten, den Sprints, abgearbeitet. Der Input für jeden Sprint sind die Ergebnisse des letzten Sprints.

VORBEREITUNG (findet während des vorhergehenden Sprints statt):

1. Erstelle User Stories inkl. Akzeptanzkriterien im Product Backlog — POX
2. Bespreche vorbereitete User Stories — POK, Kunde

ABLAUF EINES SPRINTS: (siehe auch Abbildung A.5)

1. Bereite Sprint Backlog in der Sprint Planung vor — Team, SM, POX
2. Starte Sprint beginnend bei der User Story mit der höchsten Priorität — Team
3. Halte Daily Scrum ab — Team, SM, (POX)
4. Nehme jede User Story möglichst früh ab — Team, POX, Kunde (wenn möglich)
5. Erstelle ein auslieferbares Produktinkrement — Team
6. Halte Sprint Review und Sprint Retrospektive ab — Team, SM, POX

Tabelle A.3 zeigt die vorgesehenen Termine für die einzelnen Meetings und Workshops über den Wochenverlauf.

Die Ergebnisse jeden Sprints sind

Mo	09:30 - 13:30	Sprint Planung
Di	09:30 - 09:45	Daily Scrum*
Mi		
Do		
Fr		
Mo	12:00 - 13:00	Scrum Lunch
Di		
Mi		
Do		
Fr	09:00 - 11:00	Sprint Review
	11:00 - 13:00	Sprint Retrospektive

*Das Daily Scrum findet täglich zum gleichen Zeitpunkt statt.

Tabelle A.3: Sprint Kalender

- das auslieferbare Produktinkrement,
- eventuell nicht fertiggestellte Stories aus dem Sprint,
- eventuell schon bekannte Arbeitsinhalte für den nächsten Sprint,
- Feedback aus dem Sprint Review und
- mind. ein Verbesserungsvorschlag aus der Sprintretrospektive.

REFERENZEN: (Pichler, 2008, S. 7-8) (Schwaber, 2007, S. 115-129)

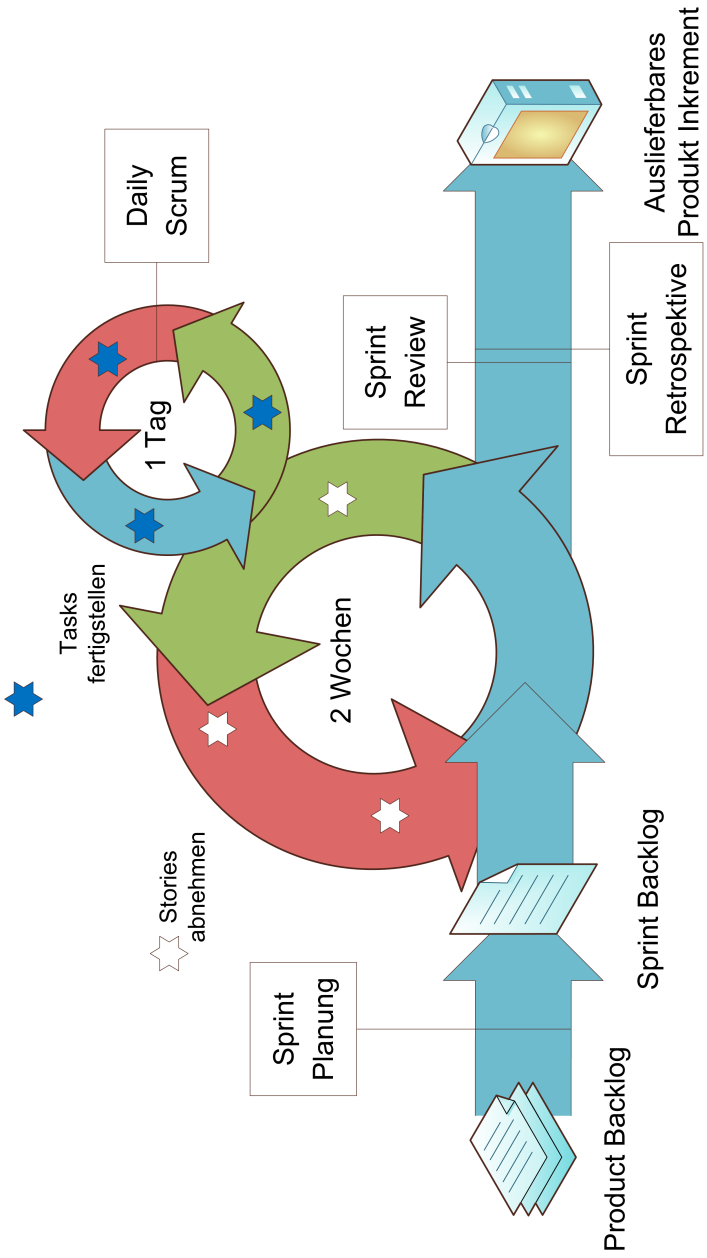

Abbildung A.5: Überblick über den Ablauf der Softwareentwicklung

A.3.6 Meetings und Workshops

Sprint Planung

In der Sprint Planung findet die Vorbereitung und Planung für den nächsten Sprint statt.

TEILNEHMER: Team, SM, POX, (POK)

DAUER: 4h (oder 2h pro Sprint-Woche)

VORBEREITUNG:

1. Bereite Ergebnisse aus dem Sprint Review auf	Team, POX, POK
2. Berechne eigene Kapazität für den nächsten Sprint	jedes Team-Mitglied
3. Stelle eventuell nicht fertiggestellte Arbeit aus vorhergehendem Sprint fest	Team
4. Stimme Sprintziel zwischen den unterschiedlichen Projekten ab	POX, POK
5. Bereite Stories inkl. Akzeptanzkriterien vor	POX, POK
6. Bestimme Fokuskapazität (=Teamkapazität x Fokusfaktor) des Teams	SM

ABLAUF:

1. Berichte eventuelle Neuigkeiten	POX, POK
2. Verhandle Sprint Ziel	Team, POX ca. 20 min
3. Plane Feedback aus dem Sprint Review ein	Team, POX, POK
4. Plane nicht fertiggestellte Arbeit aus vorhergehendem Sprint ein	Team, POX, POK
5. Wähle Stories für den Sprint aus	Team, POX, POK ca. 40 min Pause 10 - 20min
6. Definiere Tasks zu den Stories	Team
7. Definiere eventuelle Tech Tasks	Team
8. Plane gefundene Bugs ein	Team ca. 90 min
9. Bestimme endgültiges Commitment	Team
10. (*optional*) Lege Termine für den nächsten Sprint fest (z.B. Änderung des Daily Scrum)	Team, SM ca. 20 min
11. Bilde Sprint-Backlog nach Prioritäten und Abhängigkeiten am Sprintboard ab	Team, SM, (POK, POX)

ca. 10 min

Die Agenda ist durch eventuell anfallende Besonderheiten des jeweiligen Sprints zu ergänzen. Im Teil I (bis zur Pause) der Sprint-Planung sind die jeweiligen Product Owner erforderlich, Teil II kann vom Team zusammen mit dem SM abgehalten.

Das **Ergebnis** einer Sprint Planung ist immer das vollständige Sprint-Backlog inklusive Prioritäten und Abhängigkeiten. Findet kein Daily Scrum vor dem Beginn der Arbeiten am Sprint-Backlog statt, wählen die Team-Mitglieder im Anschluss an die Sprint Planung Tasks zur Bearbeitung aus. Ein weiteres Ergebnis der Sprint Planung sind, wenn notwendig, Terminfestlegungen für Meetings und Workshops. Dies beinhaltet auch spezielle von der jeweiligen Entwicklungssituation abhängige Termine wie z.B. ein Integrationsworkshop.

REFERENZEN: (Pichler, 2008, S. 51, 87-102) (Cohn, 2006, S. 145-166) (Elssamadisy, 2009, S. 77-80)

Daily Scrum

Im Daily Scrum wird die tägliche Arbeit geplant und Abhängigkeiten zwischen den Tasks, den Teammitgliedern und der noch zur Verfügung stehenden Teamkapazität berücksichtigt. Es findet im Stehen vor dem Sprintboard statt.

TEILNEHMER: Team, SM, (optional) PO

DAUER: max. 15 min

VORBEREITUNG:

Aktualisiere Trac und Sprintboard — Team

ABLAUF:

1. Beantworte die folgenden 3 Fragen — Jedes Teammitglied
   - Was habe ich seit dem letzten Daily Scrum fertiggestellt?
   - Was habe ich vor bis zum nächsten Daily Scrum zu machen?
   - Was blockiert mich bei meiner Arbeit?
2. Kontrolliere Sprint Burndown Chart und Ergreife eventuell notwendige Maßnahmen — Team, (PO)
3. Bespreche Allfälliges — Team, SM, (PO)

Der SM sollte besonders auf Frage 3 achten, da die dort genannten Blockaden einem Arbeitsauftrag an ihn entsprechen.

Die **Ergebnisse** des Daily Scrums sind

- der weitere Arbeitsplan, zu dem jedes Teammitglied sich verpflichtet,
- das angepasste Sprint Backlog und
- ein oder mehrere Blockaden (Impediments) zur Entfernung durch den SM.

REFERENZEN: (Pichler, 2008, S. 52, 104-107) (Yip, 2006) (Sutherland, 2006) (Elssamadisy, 2009, S. 93-98)

Sprint Review

Im Sprint Review stellt das Team das im letzten Sprint fertiggestellte Produktinkrement den Stakeholdern (PO, Kunde, Benutzer, Manager, Marketing- und Vertriebsmitarbeiter, etc.) vor. Der SM moderiert das Sprint Review.

TEILNEHMER: Team, SM, POK, POX, Kunde XiTrust, verschiedenste Stakeholder

DAUER: max. 2h

VORBEREITUNG: Da das Produkt Inkrement nach der Checkliste für Fertigstellung von Stories bereits während des Sprints demonstrierbar gemacht werden muss, ist die Vorbereitung auf ein Minimum zu beschränken (z.B. Vorbereiten des Demonstrationsraums).

ABLAUF:

1. Stelle das Sprint-Ziel kurz vor	Team
2. Demonstriere jede abgeschlossene User Story live	Team
3. Probiere das Inkrement selbst aus	Stakeholder
4. Notiere Feedback der Stakeholder	Team, PO
5. (*optional*) Bespreche eventuell nicht fertig gestellte Stories, Probleme, aber auch positive Ereignisse	Alle
6. Entscheide, ob das Inkrement abgenommen ist	PO

Die Live-Demonstration ist immer mit dem letzen offiziellen Build, das auf einem System möglichst nahe der Produktivumgebung installiert ist, durchzuführen.

Die **Ergebnisse** des Sprint Reviews sind

- im nächsten Sprint zu behebende Fehler,
- Verbesserungsvorschläge vom Kunden und
- eventuell neue Prioritäten für den nächsten Sprint.

REFERENZEN: (Pichler, 2008, S. 107-111) (Elssamadisy, 2009, S. 103-107)

Sprint Retrospektive

In der Sprint Retrospektive wird über einen oder mehrere vergangene Sprints reflektiert und Verbesserungsexperimente für den nächsten Sprint geplant. Der SM moderiert die Retrospektive und bleibt daher neutral.

TEILNEHMER: Team, SM, POX, (POK)

DAUER: max. 2h

VORBEREITUNG:

1. Bereite Thema, Aktivitäten und Materialen für die Aktivitäten vor	SM
2. Bereite Daten zur Analyse vor (z.B. Anzahl der fehlgeschlagenen Builds, wiederkehrende Sprint Burndown Muster)	SM
3. Verifiziere Ergebnisse des letzten Experiments	SM, Team

ABLAUF:

1. Begrüße das Team	SM
2. Bespreche die Ergebnisse des letzten Experiments	Team, POX
3. Entscheide, ob das Experiment Teil des Prozesses wird (War es erfolgreich?)	Team, POX
4. Stelle das Thema der Retrospektive vor	SM
5. Sammle Daten zum Thema	Team, POX
6. Forsche nach Ursachen für und generiere Einsichten über die gesammelten Daten	Team, POX
7. Entscheide über Maßnahmen und Experimente	Team, POX
8. Bewerte und schließe Retrospektive ab	Team, POX, SM

Eine Retrospektive ist immer ein moderierter Workshop.

Das **Ergebnis** der Retrospektive ist ein Experiment, das im nächsten Sprint durchgeführt wird. Weiters wird das letzte Experiment zur Übernahme in die Prozesse angenommen oder abgelehnt. Wird ein Experiment angenommen ist das Team dafür verantwortlich, dass die Prozessbeschreibung entsprechend geändert werden.

VORLAGE: svn:/Vorlagen/Projektmanagement/2009-06 Agenda Retrospektive.dotx

REFERENZEN:

- Retrospektive: (Pichler, 2008, S. 111-115) (Derby and Larsen, 2006) (Carr, 2008a) (Carr, 2008b) (Elssamadisy, 2009, S. 109-113)
- Workshop: (Elssamadisy, 2009, S. 245-248)

Anforderungsworkshop

Der Anforderungsworkshop findet unregelmäßig statt, meist zu Beginn eines Releases, Projekts, bei großen Veränderungen der Anforderungen oder zur Verfeinerung von existierenden Anforderungen.

TEILNEHMER: Team, SM, PO, weitere Stakeholder

DAUER: je nach Menge der Anforderungen, max. 4h

VORBEREITUNG:

Erstelle eine Produktmission	PO, Kunde
oder	
Beschreibe Art der Veränderung	PO, Kunde
oder	
Bereite Anforderungen als User Story im Produkt-Backlog für Workshop vor	PO, Kunde

ABLAUF:

1. Bespreche jede User Story	PO, Kunde, Team
2. Schätze jede Story mittels Planning Poker	Team
3. Weise jeder Story eine initiale Priorität zu	PO, Kunde

Das Ergebnis des Anforderungsworkshop sind Anforderungen in Form von priorisierten, geschätzten User Stories.

REFERENZEN:

- Anforderungsworkshop: (Pichler, 2008, S. 37-40)
- Schätzen: (Cohn, 2006, S. 33-75)
- Planning Poker: (Cohn, 2006, S. 56-59) (Elssamadisy, 2009, S. 87-91)

Scrum Lunch

Der Scrum Lunch ist ein vom Team gemeinsam eingenommenes Mittagessen. Er findet alternierend zur Planungssitzung statt. Dort werden allfällige, für das Team relevante Themen besprochen, z.B. besondere Entwicklungspraktiken, Programmiertechniken oder Prozessfragen.

TEILNEHMER: Team, SM, (POX)

DAUER: max. 1h

VORBEREITUNG:

1. Bestimme ein Teammitglied zum Essenslieferanten oder Koch	Team, SM
2. Bestimme ein Thema für den Scrum Lunch	Team, SM

3. Bereite Informationen zu dem Thema vor — Team, SM

ABLAUF: Der Scrum Lunch hat keinen geregelten Ablauf.

Das **Ergebnis** des Scrum Lunch ist erweitertes Wissen über das besprochene Thema.

A.3.7 Entwicklungspraktiken

Checkliste für Fertigstellung (Definition of Done)

Die Checkliste für die Fertigstellung definiert klar und eindeutig, wann ein Arbeitsauftrag wirklich fertig ist. Ist der Arbeitsauftrag nach dieser Checkliste abgeschlossen, sollte das Ergebnis auslieferbarer Software so nahe wie irgend möglich sein. Für jede Entwicklungsebene gibt es eine eigene Checkliste.

REFERENZEN: (Elssamadisy, 2009, S. 99-102) (Schwaber, 2007, S. 133)

Task

1. Task ist vollständig implementiert, getestet und dokumentiert
   - Das Code Format entspricht dem Code Format des Projekts (settings-Folder im Eclipse)
   - Alle Variablen/Methoden haben sprechende Namen (keine temps, pimfps, etc.)
   - Statt Inline-Kommentaren sind private Hilfsmethoden verwendet und mit einem sprechenden Namen versehen. In Eclipse kann dazu ganz einfach die 'Refactor → Extract Method' Funktion verwendet werden.
   - Alle Return Values sind kommentiert, vor allem bei Collections.
   - Jedes Interface ist so vollständig dokumentiert, dass auch ein Anderer die Implementierung einfach durchführen kann.
   - Jeder Entwickler entscheidet, basierend auf der Wichtigkeit des Codeteiles für ein Kunden- oder Core-Bundle, ausführlicher zu dokumentieren (Parameter, Zweck etc.)
   - Die Funktionalität ist durch entsprechende (Unit) Tests überprüft, dabei werden realitätsnahe Daten (echte Namen, etc.) verwendet
2. Alle Tests wurden bestanden
   - Unit Tests
   - GUI Tests
3. Alle Code Teile sind integriert
   - Der eingecheckte Code ist sauberer als zum Zeitpunkt des Auscheckens (Boy-Scout-Regel)
   - Vor dem Einchecken wurden alle Änderungen im Team-Synchronisations-Modus überprüft und nur notwendige Teile eingecheckt.
   - SVN Commit ist durchgeführt.

- Es wurde ein Build im Hudson gestartet (für den e3server manuell) und kontrolliert, ob alles gutgegangen ist.

4. Sprintboard und Trac ist aktualisiert
    - Die wichtigsten, während der Taskbearbeitung getroffenen Entscheidungen sind im Trac-Ticket dokumentiert.
    - Ticket und Kärtchen sind im Status 'Done'.

Story

1. Alle Tasks einer Story sind nach deren Checkliste für die Fertigstellung fertig
2. Es existieren keine bekannten Bugs
3. Die notwendige Dokumentation (z.B. Betriebshandbuch, Benutzerhandbuch) zur Story existiert
4. Es existiert ein Build, das die Story beinhaltet
5. Die Story ist bereit für eine Präsentation
    - Das Build ist am Testsystem installiert
    - Innerhalb von (max.) 30 min kann die Story dem Kunden präsentiert werden
6. Die Story ist vom PO abgenommen

Sprint

1. Es existiert ein Build mit allen in diesem Sprint fertiggestellten Stories
2. Das offizielle Build ist beim Kunden installiert
3. Die Live-Demonstration mit dem Kunden wurde durchgeführt und das Feedback erfasst
4. Sprintboard und Trac sind für die nächste Sprintplanung vorbereitet

Release

1. Es existieren keine bekannten Bugs

Projekt

1. Ein Build für das Projekt existiert und wurde für den Support dokumentiert (siehe auch A.3.8)
2. Alle Kundentests wurden bestanden
3. Das Build für das Projekt ist beim Kunden in der Produktivumgebung im Einsatz
4. Der Kunde ist mit der Lösung (dem Software Produkt) einverstanden
    - Der Kunde hat die Möglichkeit die Software als vollständig zu bewerten, auch wenn noch nicht alle zu Beginn besprochenen Stories umgesetzt wurden

Continuous Integration

Continuous Integration verhindert das Auseinandertriften verschiedener Codebasen, die durch das Auschecken durch Teammitglieder oder Erstellen eines Branches entstehen. Dazu ist folgendes notwendig:

Configuration Management Das Configuration Management System (CMS) sorgt dafür, dass die Teammitglieder unabhängig voneinander an allen Teilen des Codes arbeiten können. Überschneiden sich Änderungen, wird dies vom CMS angezeigt und können so behoben werden, ohne dass die Änderung eines der Teammitglieder verloren geht. Jede Änderung wird außerdem versioniert, sodass im Problemfall sofort auf eine frühere Version zurück gestellt werden kann. Weiters können Codeteile und Versionen so zusammengefasst werden, dass sie eine Produktversion, ein Release oder einen Patch identifizieren.

XiTrust verwendet SubVersion (SVN) (CollabNet, b) für genannte Zwecke. Jedes Team-Mitglied integriert die Änderungen an der eigenen Codebasis so oft wie möglich in die im SVN abgelegte, gemeinsame Codebasis. So oft wie möglich bedeutet nach ein paar Stunden, wenn ein Task fertig ist, spätestens jedoch am Ende des Tages.

Build System Das Build System wandelt den Code in ausführbare Software um. Außerdem führt es automatisch eine Reihe von Tests am Code und an der ausführbaren Software durch. XiTrust verwendet Hudson (CollabNet, a) als Build System. Dort muss für jedes Softwareprodukt ein Build definiert sein, der manuell oder automatisch bei Änderungen gestartet werden kann.

Jedes Teammitglied ist dafür verantwortlich, seine Änderungen unverzüglich im Build zu integrieren. Dadurch sind eventuelle unerwünschte Effekte der Änderungen auf andere Teile der Software sofort erkennbar und können unmittelbar behoben werden. Das Teammitglied darf nicht länger als ein paar Minuten auf das Erstellen des Builds warten müssen. Andernfalls besteht die Gefahr, dass das Erstellen eines Builds vernachlässigt wird. Länger andauernde Tests sollten daher nur im automatischen, nächtlichen Build durchgeführt werden.

REFERENZEN: (Beck, 2000, S 97-99), (Poppendieck and Poppendieck, 2007, S. 202-203)

Test Driven Development (TDD)

Das Testen der Software hat einen sehr hohen Stellenwert in der agilen Softwareentwicklung. Nur durch diszipliniertes Testen ist gewährleistet, dass jederzeit die gesamte Software geändert werden kann, ohne dass sie dadurch instabil wird. Eine bewährte Praktik dafür ist Test Driven Development. Dabei werden die Testfälle vor dem eigentlichen Code erstellt und somit ist gewährleistet, dass zu jedem Codefragement auch

Tests existieren. Gleichzeitig wird der Code leichter testbar. TDD kann auf mehreren Ebenen verwendet werden.

Auf der Ebene der Komponenten- und Integrationstests werden JUnit-Tests (Beck and Saff, 2009) verwendet. Ist im Softwareprodukt eine GUI enthalten, muss darauf geachtet werden, diese möglichst dünn zu halten. Unit Tests testen dann die Einsprungspunkte der GUI. Dies wird oft als 'Testen direkt unter der GUI' bezeichnet.

Auch auf der Ebene der Akzeptanztests kann TDD angewendet werden. Der Product Owner und Kunde spezifizieren ihre Akzeptanzkriterien als automatisch ausführbare Tests. Entsprechende Werkzeuge sind noch in die Entwicklungsinfrastrukur bei XiTrust zu integrieren. Vorstellbar wäre das Spezifizieren der Akzeptanzkriterien mittels FitNesse (Murgridge and Cunningham, 2009). Dies könnte auch für das Erstellen einer ausführbaren Spezifikation verwendet werden, sollte ein Projekt es erfordern, eine Vorabspezifikation zu erstellen.

REFERENZEN: (Beck, 2000, S. 115-119)

Coding Standards

Coding Standards unterstützen die Zusammenarbeit aller Teammitglieder, da sie die Verständlichkeit und Wartbarkeit des Codes erhöhen. Sie beschreiben, wie verschiedene Aspekte des Codes auszusehen haben. Dazu zählen Formatierungsoptionen wie z.B. Klammernsetzung, Leerzeilen und Einrückung, aber auch die Namensgebung von Variablen, Methoden und anderen Objekten. Weiters zählen dazu die Verwendung von Kommentaren und Klassendesigns. Coding Standards müssen so weit als möglich automatisch angewandt und überprüft werden.

Die Coding Standards für die Codeformatierung sind bei XiTrust im .settings-Ordner eines jeden Eclipse-Projekts abgelegt. Sie werden damit automatisch auf neuen Code angewandt. Für die automatisierte Überprüfung der anderen Aspekte eines Coding Standards wird der Einsatz von CheckStyle (Burn, 2009) oder einem ähnlichen Werkzeug empfohlen.

REFERENZEN: (Beck, 2000, S. 61, 69) (Poppendieck and Poppendieck, 2007, S. 193-194)

Collocated Team

Der Begriff *Collocated Team* bedeutet, dass das Scrum Team seine Arbeit in einem gemeinsamen Raum verrichtet. Alle Teammitglieder sind dort unterzubringen. Trotzdem muss jedem Mitarbeiter noch genug persönlicher Raum zur Verfügung stehen. Ist der Raum nicht groß genug, muss zumindest ein Teil des Raumes als Rückzugsmöglichkeit gestaltet sein. Im selben Raum sind auch das Sprintboard und andere Informationsquellen anzubringen.

REFERENZEN: (Beck, 2000, S. 77-80) (Poppendieck and Poppendieck, 2007, S. 211-213)

Minimierung der Codezeilen

Jede geschriebene Zeile Code muss getestet und gewartet werden. Jede Zeile Code, die für die aktuelle Aufgabe nicht notwendig ist, muss folglich auch getestet und gewartet werden. Der Aufwand für das reine Schreiben von zusätzlichem Code wird von dem tatsächlichen, über den Lauf der Zeit kumulierten Aufwand für diesen Code oft um ein Vielfaches überschritten. Letzterer enthält neben den initialen Tests zusätzlich den Aufwand für das wiederholte Lesen, Anpassen von Tests, Refactoring und ähnlichen Tätigkeiten.

Wird der Codeteil später tatsächlich benötigt, wird er meist auch noch einmal überarbeitet. Daher ist es kostengünstiger den geschriebenen Code auf jene Teile zu beschränken, die für die aktuell vorliegende Aufgabe notwendig sind. Dies trifft aus obengenannten Gründen auch zu, wenn der zu erwartende Aufwand später höher ist als jetzt.

REFERENZEN: (Poppendieck and Poppendieck, 2007, S. 75 u.a.) (Beck, 2000, S. 103-113)

Code Reviews

Das Durchführen von Code Reviews ist eine Praktik, die der internen Weiterbildung der Mitarbeiter dient. Der Code eines Mitarbeiters wird auf Komplexität, Wiederholungen, mögliche Probleme bei Änderungen und andere unerwünschte Eigenschaften untersucht. Jeder Mitarbeiter kann einen Code Review mit einem anderen Mitarbeiter anfordern. Dieser wird dann gemeinsam mit einem erfahrenen Mitarbeiter und anderen interessierten Personen durchgeführt.

Code Reviews sind kein Werkzeug für die Fehlersuche oder das Einhalten von Coding Standards. Ersteres wird durch ausreichend Tests auf den verschiedenen Ebenen abgedeckt. Zweiteres sollte automatisch von Eclipse und Code-Analyse-Werkzeugen erledigt werden.

REFERENZEN: (Poppendieck and Poppendieck, 2007, S. 194-195)

Dokumentation

Es sind zwei Arten von Dokumentation zu unterscheiden:

Bestellte Dokumentation. Vom Kunden bestellte Dokumentation ist Teil des Softwareprodukts und muss daher im entsprechenden Ausmaß auch im Produktinkrement enthalten sein.

Eigene Dokumentation. Es gilt das Prinzip, dass mündliche Kommunikation und ausführbare Dokumentation der schriftlichen Dokumentation vorzuziehen ist. Ausführbare Dokumentation ist in erster Linie der Code selbst. Dieser sollte

möglichst selbsterklärend sein und mit wenig zusätzlicher Dokumentation auskommen. Weiters zählen dazu die schon genannten Unit-Tests und Akzeptanztests. Letztere können auch mit erklärendem Text versehen werden.

Ist noch zusätzliche schriftliche Dokumentation erforderlich, sind Diagramme und grafische Darstellungen reinem Text vorzuziehen. Die Dokumentation zu Entscheidungen im Laufe der Projektarbeit ist, wie unter A.3.7 beschrieben, im Trac zu finden.

A.3.8 Release

Ein Release entspricht einem Produktinkrement, das eine vorher festgelegte Menge von Anforderungen implementiert. Es entsteht über einen oder mehrere Sprints. Im Fall von XiTrust setzt sich die Menge Anforderungen für ein Release aus Anforderungen aus den Bereichen Projekt, Support und interne Entwicklung zusammen. Wieviele Anforderungen aus welchen Bereichen ein Release ausmachen, muss entsprechend geplant werden.

Je nach Art der Anforderungen, die in einem Release umgesetzt werden, ist zu unterscheiden zwischen Minor und Major-Releases.

Major-Release. Ein Major-Release enthält eine signifikante Änderung des Softwareprodukts und damit auch eine größere Menge an umgesetzten Anforderungen. Es wird über einen längeren Zeitraum (z.B. ein Halbjahr) entwickelt. Das Ziel für ein Major-Release wird daher als Release-Mission (siehe A.3.2) formuliert.

Ein Major-Release wird bei XiTrust als internes Entwicklungsprojekt mit abgewickelt. Auftraggeber ist dabei der interne Kunde. Er wird vom POX repräsentiert. Wird im weiteren Text von Projekt gesprochen, ist damit auch die interne Entwicklung gemeint.

Minor-Release. Ein Minor-Release enthält eine kleine Menge an Anforderungen und wird über den Zeitraum von ein oder zwei Sprints entwickelt. Es ist daher relativ genau planbar. Die als Story beschriebenen Anforderungen für jeden Sprint eines Minor-Releases sind vom POX wie folgt auszuwählen:

1. Selektiere Stories mit der höchsten Priorität aus dem Support.
2. Selektiere Stories aus dem Projekt mit der höchsten Priorität.
3. Führe die Sprint Planung (A.3.6) mit der Menge der vorselektierten Stories durch.
4. Nominiere mindestens eine Story als potentiell optional. Diese kann aus dem Sprint entfernt werden, sollten dringende Supportanfragen (siehe A.5) zu lösen sein.

Das Priorisieren der Supporttätigkeiten ist von XiTrust zu entwickeln. Die Projekte können wie folgt priorisiert werden:

1. Reihe die laufenden Projekte nach geplantem Fertigstellungstermin des nächsten Releases, früheste zuerst.
2. Aus den zuerst gereihten wähle maximal 3 Projekte zur Bearbeitung in der genannten Reihenfolge
   (a) Der zu erwartende Fertigstellungstermin liegt nach dem geplanten Fertigstellungstermin
   (b) Der zu erwartende Fertigstellungstermin entspricht dem geplanten Fertigstellungstermin
   (c) Der zu erwartende Fertigstellungstermin liegt vor dem geplanten Fertigstellungstermin

Erstellen

Der notwendige Ablauf beim Erstellen eines Releases ist von der XiTrust Entwicklung zu definieren.

Jedes Release muss eine eindeutige Versionsnummer erhalten. Dies dient vor allem der Identifizierung eines Codepaketes im SVN, aber auch der Dokumentation, welche exakte Version bei den Kunden in Verwendung ist. Die Versionsnummer wird zusammengesetzt aus

$$< majorrelease > . < minorrelease > . < patch > - < build > . \qquad \text{(A.1)}$$

Die Nummern werden wie folgt bestimmt:

Major-Release. Erhöhe um 1, wenn ein Major-Release erstellt wird.

Minor-Release. Erhöhe um 1, wenn ein Minor-Release erstellt wird.

Patch. Erhöhe um 1, wenn ein Hotfix erstellt wird und direkt an den Kunden ausgeliefert wird. Setze auf 0, wenn das Minor-Release erstellt wird, das alle Hotfixes, die während der Releaseentwicklung entstanden sind, enthalten muss. Patch Nummer werden daher nur zwischen den Minor-Releases hochgezählt.

Build. Die Buildnummer entspricht der Nummer des Build des Releases im Hudson.

Installation

Bei der Installation eines Releases wird zwischen einer Neuinstallation und einem Update unterschieden. Es ist zu erwarten, dass die Neuinstallation und das Update gemeinsame aber auch unterschiedliche Schritte erfordern. Die genaue Abwicklung der Installationsarten ist von der XiTrust Entwicklung und dem Support zu definieren.

A.3.9 Produkt Roadmap

Die Produkt Roadmap dient der Steuerung aller Entwicklungsanstrengungen. Sie wird vom Kunden XiTrust zusammen mit den POs erstellt. Sie listet zukünftige, bereits geplante aber auch vom Kunden XiTrust erwünschte neue Funktionen auf. Damit können einerseits aus einer Reihe von möglichen Projekten jene ausgewählt werden, die das Produkt in die gewünschte Richtung entwickeln. Anderseits können aktiv Kunden gewonnen werden, die diese Funktionen nutzenbringend einsetzen können.

A.3.10 Speziell zu berücksichtigende Situationen

Softwareentwicklung mit Diplomanden

Bei XiTrust werden Teile der Produktlinie von Diplomanden entwickelt. Diese können daher nicht vollständig in das Entwicklungsteam integriert werden, da sie nur die Entwicklung gewisser Funktionen übernehmen können, bestimmt durch das Thema der jeweiligen Diplomarbeit. Dennoch darf die Entwicklung mit Diplomanden nicht vollständig ungesteuert bleiben.

Eine Diplomarbeit wird daher vom Diplomanden zusammen mit dem POX wie ein internes Entwicklungsprojekt geplant. Dazu wird ein initiales Produkt-Backlog erstellt (siehe A.4.2) und sukzessive verfeinert und priorisiert. Die Granularität der Stories hängt einerseits von der Softwareentwicklungserfahrung des Diplomanden und andererseits von seiner Fähigkeit selbstständig zu arbeiten ab. Der Arbeitsaufwand einer Story darf die Sprintlänge nicht überschreiten.

Jeder Diplomand wählt zusammen mit dem POX eine Sprintlänge und den Sprintbeginn. Die Sprints sind wie bereits beschrieben abzuwickeln, mit der Ausnahme von folgenden Unterschieden:

- Es findet kein Scrum Lunch statt. Der Diplomand kann am Team Scrum Lunch teilnehmen, um vom dort generierten Wissen zu profitieren.
- Es findet keine eigene Retrospektive statt. Verbesserungen im Umgang mit Diplomanden werden in der unternehmensweiten Retrospektive oder bei Bedarf direkt mit dem Diplomanden besprochen.
- Teil II der Sprint Planung wird vom Diplomand selbstständig und bei Bedarf durchgeführt. Dies bedeutet, er definiert und priorisiert die notwendigen Tasks zu den geplanten Stories selbst.
- Das Backlog für Diplomanden kann in einer eigenen Trac-Umgebung verwaltet werden. Verbindungen zum Trac der Entwicklung können über Inter-Trac-Links hergestellt werden.

Der Diplomand muss weiters die gleichen Entwicklungspraktiken (siehe A.3.7) wie das Scrum Team anwenden. Andernfalls besteht die Gefahr, dass das fertige Produkt

nicht der erwarteten oder benötigten Qualität entspricht. Er ist daher vor Beginn der Codierungsphase mit diesen von einem erfahrenen Entwicklungsmitarbeiter vertraut zu machen.

Jeder Diplomand wird daher von zwei XiTrust-Mitarbeitern betreut. Das ist einerseits der POX, der die Kundensicht liefert. Anderseits wird er von einem Entwicklungsmitarbeiter bei technischen Problemen und entwicklungsrelevanten Fragen unterstützt. Dieser führt auch notwendige Code Reviews mit ihm durch. Aufgrund der Tatsache, dass die Betreuung eines Diplomanden erhöhten Zeitbedarf und somit Einbußen bei der Entwicklungsgeschwindigkeit für den jeweiligen Entwicklungsmitarbeiter und POX bedeutet, wird empfohlen, nicht mehr als einen Diplomanden pro Scrum Team einzusetzen.

Softwareentwicklung mit Lehrlingen oder unerfahrenen Mitarbeitern

Agile Softwareentwicklung mit unerfahrenen Mitarbeitern ist eigentlich nicht empfohlen (Boehm and Turner, 2003). Ist dies trotzdem erwünscht und notwendig, muss eine Trainingsstrategie für den entsprechenden Mitarbeiter entworfen werden. Das Ziel einer solchen Strategie ist, dass der Mitarbeiter möglichst bald selbstständig zur Entwicklung beitragen kann.

Eine solche Strategie enthält folgende Elemente:

- Einweisung in die internen Prinzipien und Praktiken
- Aufbau und Weiterentwicklung von Wissen über die agile Softwareentwicklung
- Aufbau und Weiterentwicklung von technischem Wissen
- Aufbau und Weiterentwicklung der notwendigen Soft Skills wie Selbstständigkeit, Selbstorganisation, Zeitmanagement und Teamarbeit

Diese Trainingsstrategie ist als Mix aus interner und externer Schulung zu gestalten. Sie muss immer speziell auf den entsprechenden Mitarbeiter abgestimmt werden.

A.4 Projektabwicklung

A.4.1 Einleitung

Dieser Abschnitt beschreibt die wichtigsten Abläufe und Aufgaben bei der Abwicklung eines Projekts bei XiTrust. Wie der Abschnitt Software Entwicklung (A.3) ist er eine Schnellreferenz. Weiterführende Informationen können der angegebenen Literatur entnommen werden.

Voraussetzung ist Basiswissen in der Projektabwicklung und die Anwendung entsprechender Werkzeuge. Weiters wird folgende Literatur empfohlen:

- Goal Directed Project Management (Andersen et al., 2004)

Die Abwicklung eines Projekts wird in drei verschiedene Phasen eingeteilt: Projektvorbereitung (A.4.2), Projektumsetzung (A.4.4) und Projektabschluss (A.4.5). Die Projektplanung A.4.3 ist schwerpunktmäßig in der Projektvorbereitung anzusiedeln, jedoch ist sie auch während der Umsetzung und des Abschließens immer wieder anzupassen und zu erweitern. In jeder Phase sind unterschiedliche Aufgaben zu erfüllen. Verantwortlich für die ordnungsgemäße Abwicklung eines Projekts ist der Product Owner Kunde (POK). Er plant und steuert das Projekt in Zusammenarbeit mit dem POX, dem Team und dem Projekteigentümer beim Kunden.

Abbildung A.6 zeigt den gesamten Projektprozess bei XiTrust.

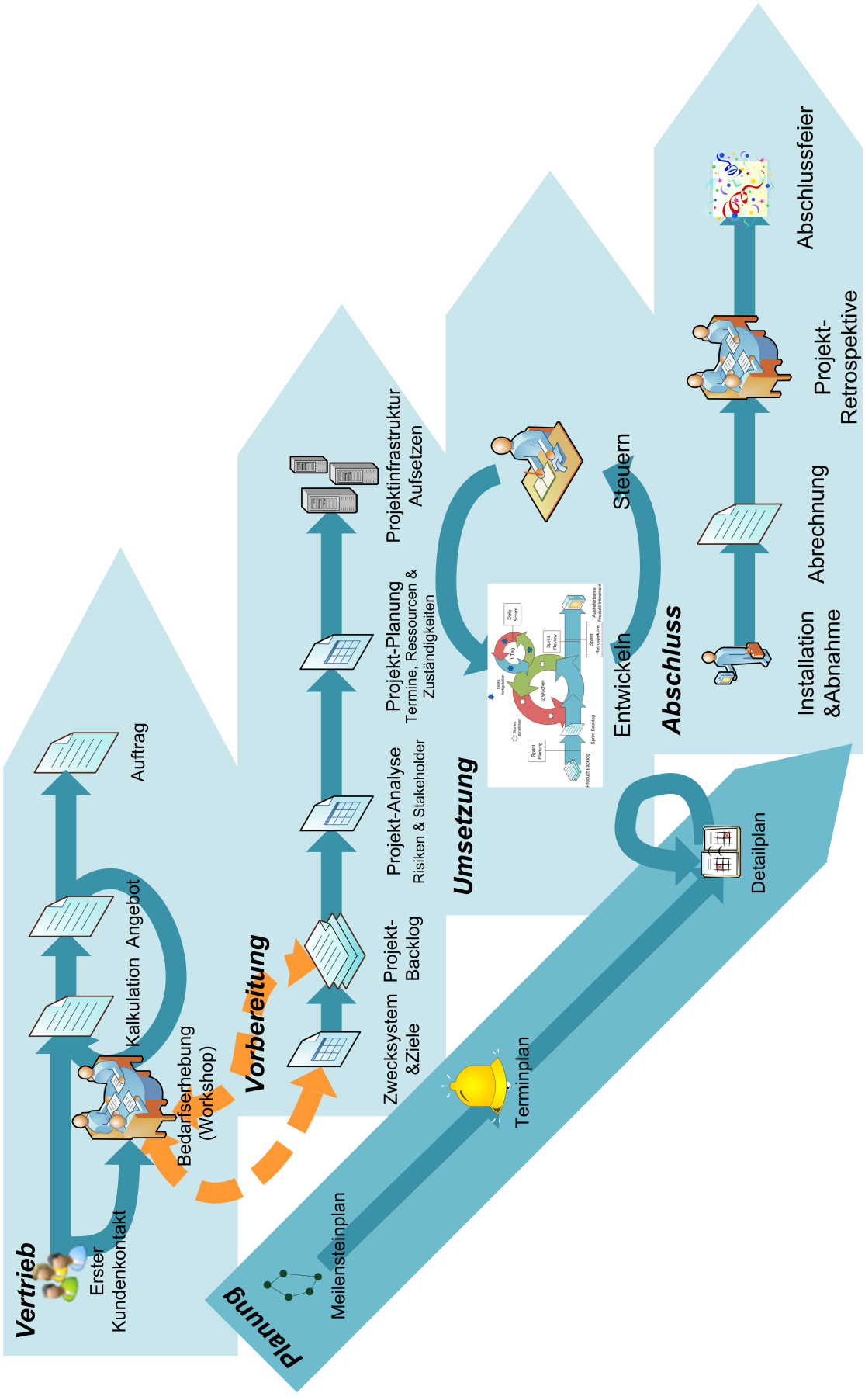

Abbildung A.6: Der Projektprozess

A.4.2 Projektvorbereitung

Gerade bei der Anwendung von agiler Softwareentwicklung ist eine gute Projektvorbereitung besonders wertvoll. Sie legt die Basis für eine gute und erfolgreiche Zusammenarbeit mit dem Kunden, Lieferanten, Endbenutzern und anderen Stakeholdern. Folgende Tätigkeiten sind empfohlen.

1. Definiere ein Zwecksystem und Ziele	POK, Kunde
2. Identifiziere und analysiere alle Stakeholder	POK, (Kunde)
3. Erstelle und schätze das Produkt-Backlog	POK, Team, SM
4. Erstelle den Projektplan	POK, Kunde
5. Definiere interne und externe Verantwortlichkeiten	POK, Kunde, Lieferanten
6. Identifiziere und plane Maßnahmen gegen eventuelle Risiken	POK, (Kunde, Lieferanten)
7. Bereite die Projektinfrastruktur vor	Team, SM, (POK)

Die folgenden Abschnitte geben eine genauere Beschreibung der einzelnen Aufgaben. Eine Vorlage für die schriftliche Ausarbeitung dieser Aufgaben ist im SVN ablegt (siehe A.4.2).

Zwecksystem und Ziele

Das Zwecksystem und die Ziele bilden die Grundlage für jedes Projekt. Der Projektzweck beschreibt die vom Unternehmen, in der Regel ein Kunde von XiTrust, erwünschte zukünftige Situation. Ein Ziel ist eine geplante Projektlieferung oder -leistung. Besonders bei einem agilen Entwicklungsvorgehen ist es wichtig, sich genügend Zeit dafür zu nehmen, da spätere Anforderungen anhand des hier definierten Projektzwecks bewertet werden können. Weiters ist ein gut definiertes Zwecksystem eine große Stütze bei dem Formulieren der Produktmission und des Scopes.

Die Vorgehensweise für das Erstellen eines Zwecksystems ist:

1. Formuliere einen Hauptzweck
2. Zerlege den Hauptzweck in mehrere detailierte Subzwecke, die den Hauptzweck verfeinern
3. Wiederhole 2. mehrmals für jeden Subzweck, je nach Komplexität und Größe des geplanten Projekts
4. Beschränke, wenn notwendig, das Projekt auf einen oder mehrere Subzwecke

Das Zwecksystem wird wie in Abbildung A.7 als Hierarchie ähnlich einem Organigramm abgebildet. Die dem geplanten Projekt zugeordneten Subzwecke sind dabei grau hinterlegt. Für die anderen Subzwecke könnten eigene Projekte geplant werden.

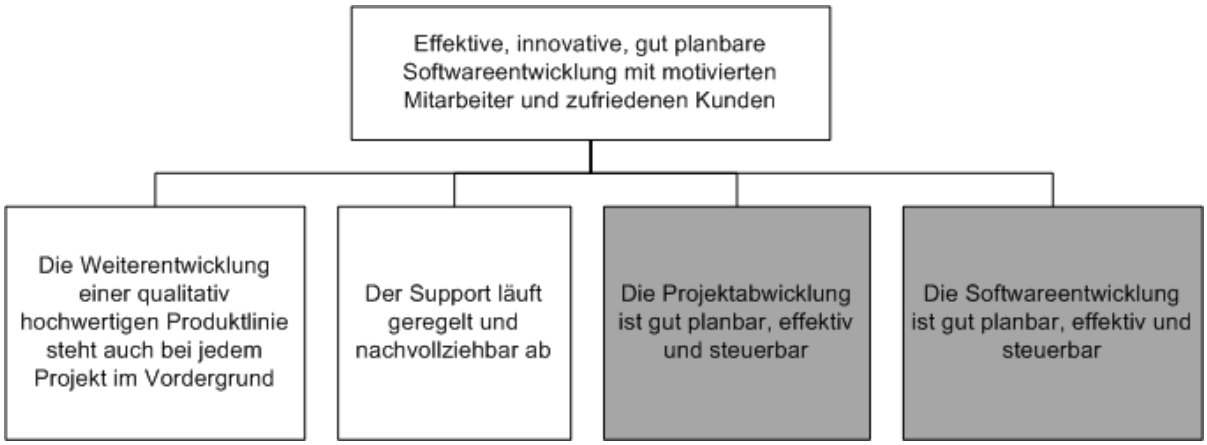

Abbildung A.7: Beispiel für ein Zwecksystem

Der erste Entwurf des Zwecksystems wird vom POK und dem Projekteigentümer erstellt. Dabei sollte immer auch die Unternehmensstrategie und der Unternehmenszweck des Unternehmens, in dessen Organisation die Projektergebnisse eingebettet werden, berücksichtigt werden. Ein Projekt erhält die meiste Legitimation, wenn es dazu beiträgt den Hauptzweck des Unternehmens besser oder einfacher zu erreichen oder die Performance der leitenden oder unterstützenden Funktionen verbessert. Ist das Projekt Teil eines Programms oder einer Projektmenge sind auch die Zwecke der anderen Projekte zu betrachten und in das Zwecksystem aufzunehmen, damit es zu keinen Überschneidungen oder Konflikten kommt.

Ist das Projekt ein Veränderungsprojekt, d.h. Abläufe und Prozesse in der Kundenorganisation werden dadurch verändert, ist darauf zu achten, den Zweck als klaren Zielzustand nicht als Problem zu formulieren . Eine reine Problemdefinition erlaubt viele Orientierungsmöglichkeit, während ein klarer zukünftiger Zustand formuliert als Zweck die Kräfte der Beteiligten bündelt (Lüsch and Zitzke, 2004, S. 21-22). Abbildung 3.10 verdeutlicht diese unterschiedlichen Ausrichtungen. Die meisten XiTrust-Projekte sind Veränderungsprojekte, da sie manuelle Abläufe auf elektronische umstellen.

Der Entwurf des Zwecksystems wird danach in einem Workshop zusammen mit den wichtigsten Stakeholdern des Projekts verfeinert und endgültig festgelegt. Dabei ist wichtig, dass jeder mit dem Ergebnis einverstanden ist, andernfalls sind spätere Konflikte vorprogrammiert. Ein Hilfsmittel dies zu erreichen sind Methoden für Konsensentscheidungen wie die Fünf-Finger-Abstimmung (siehe (5am Solutions, 2008)). Es ist die Aufgabe des POK den Kunden bestmöglich bei der Zweckformulierung zu unterstützen.

Ist das Zwecksystem definiert, können daraus die Projektziele abgeleitet werden. Dabei ist darauf zu achten, dass die Ziele nicht als Aktivität, sondern als erwartete Lieferung oder Leistung formuliert werden. In vielen Fällen wird eines der Ziele die Installation und Anpassung der XiTrust Software sein.

Abschließend ist es noch sinnvoll, einen passenden Projektnamen zu wählen, der auch den Zweck entsprechend ausdrückt. Dies kann die Unterstützung verschiedener Sta-

keholder für ein Projekt signifikant verbessern (Andersen et al., 2004, S. 53).

REFERENZEN: (Andersen et al., 2004, S. 38-48)

Stakeholder Analyse

Stakeholder sind Personen und Gruppen, die aktiv am Projekt mitarbeiten oder passiv davon betroffen sind. In jedem Projekt gibt es verschiedene Stakeholder mit unterschiedlichsten Interessen. Die Stakeholderanalyse hilft dem Projektleiter die Stakeholder genauer kennenzulernen und ihre unterschiedlichen Interessen wahrzunehmen und mit dem Projekt zu vereinbaren. Dies hilft insbesondere bei folgenden Aufgaben:

- Auswählen von Endbenutzern für die Zusammenarbeit mit dem Entwicklungsteam
- Feststellen des Kundenwissen in Bezug auf agile Entwicklung und Auswahl einer entsprechenden Projektkommunikation und -strategie
- Minimieren von negativer Einflussnahme auf das Projekt
- Verfeinern von Ziel und Zweckdefinition. Dies trifft insbesondere zu, da durch Gespräche mit den entsprechenden Stakeholdern erst Ziele aufgedeckt werden können, die andernfalls übersehen worden wären oder als versteckte Ziele die Zufriedenheit mit dem Projektergebnis negativ beeinflussen können.

Während der Stakeholderanalyse werden die Stakeholder identifiziert, ihre Beziehungen untereinander und zum Projekt analysiert sowie Strategien und Verantwortlichkeiten für die Zusammenarbeit mit ihnen festgelegt. Tabelle A.4 zeigt das Schema für die schriftliche Ausarbeitung der Stakeholder Analyse.

Stakeholder	**Interessens-Bereich**	**Beitrag**	**Erwartungen**	**Einfluss**	**Strategie**	**Verantwortlicher**
Name oder Rolle des Stakeholders	welcher Bereich des Projektes ist für diesen Stakeholder von Interesse, z.B.: ein spezielles Ziel	Beitrag zum Projekt, z.B.: Finanzierung oder Mitarbeit, aber auch negativer Beitrag möglich	Erwartungen an das Projekt	Wie groß ist sein Einfluss auf das Projekt	Wie muss mit dem Stakeholder zusammengearbeitet werden	Wer ist dafür verantwortlich, dass die Zusammenarbeit entsprechend erfolgt

Tabelle A.4: Stakeholder Analyse Matrix

Typische Stakeholder sind in (Andersen et al., 2004, S. 48) gelistet. Darunter finden sich vor allem interne Rollen, Partner und Benutzer aber auch Behörden, Medien und externe Finanziers.

REFERENZEN: (Andersen et al., 2004, S. 48-51)

Erstellen des Projekt-Backlogs

Das Projekt-Backlog ist die Basis für den Entwicklungsanteil des geplanten Projekts. Es sollte daher mit der entsprechenden Sorgfalt erstellt werden. Wie im Produkt-Backlog werden die Anforderungen eines Projekts als Stories beschrieben.

Die einzelnen, zu Projektstart noch grob gefassten Stories, können wie bisher mit entsprechenden UML-Elementen (Use Cases, Sequenz-Diagramme) verdeutlicht werden. Das Erstellen des Projekt-Backlogs kann daher Teil des schon bisher durchgeführten Bedarfserhebungs-Workshops sein. Der folgende Ablauf beschreibt welche Schritte für die Erstellung des initialen Product Backlogs durchlaufen werden müssen.

ABLAUF:

1. Beschreibe die Produktmission (siehe auch A.4.2 und A.3.2)	PO, Kunde
2. Erstelle eine thematische Einteilung im Product Backlog	PO, Kunde
3. Erstelle alle aus derzeitiger Sicht notwendigen User Stories	PO, Kunde
4. Schätze den Aufwand für jede Story in Form von Story Points	PO, Team
5. Bestimme die Priorität jeder Story	PO, Kunde

Abbildung A.8 ergänzt die Beschreibung des Ablaufs. Die Daten des Projekt-Backlogs sind auch für die Planung des Projekts wichtig.

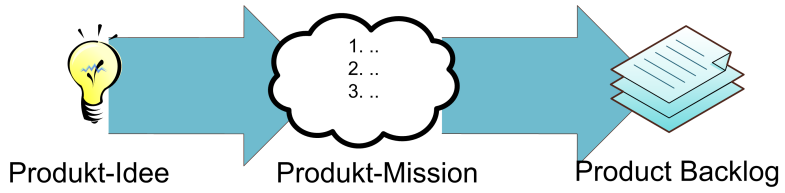

Abbildung A.8: Von der Projekt-Idee zum Projekt-Backlog

Es ist außerdem zu empfehlen, die Vorarbeit im Anforderungsbereich auf das notwendige Maß zu beschränken. Die Anforderungen sollten nicht zu früh zu detailiert aufgenommen werden, da sie sich im Verlauf der Entwicklungsarbeit noch wesentlich verändern können. Zu berücksichtigen sind dabei eventuelle vertragliche Notwendigkeiten sowie die Erfahrung des Kunden mit agilen Vorgehensweisen. Eventuell sind ihm die Vorteile eines solchen Vorgehens erst zu erläutern. Gegenseitiges Vertrauen ist dabei jedoch unabdingbar.

Bei der Menge der Anforderungen im Projekt-Backlog gilt das Credo: Weniger ist mehr. Besonders wenn das Budget beschränkt ist. Es ist besser, einige wenige Anforderungen, die den Kunden begeistern und sein Kernproblem lösen, gut umzusetzen, als viele unterschiedliche Anforderungen, die die Software zusätzlich kompliziert machen. Studien haben gezeigt, dass in vielen Softwareprodukten nur in etwa 20% der Features genutzt werden. Der Rest ist als Überproduktion anzusehen (Poppendieck and Poppendieck, 2007, S. 24).

Verantwortlichkeiten

Wie schon erwähnt, hat ein Projekt immer mehrere Beteiligte und Stakeholder. Zusätzlich zur Stakeholder Analyse ist es auch wichtig, die Verantwortlichkeiten der einzelnen Rollen oder Personen zu klären. Dies geschieht einerseits über die grundlegende Verantwortlichkeitsmatrix, die Verantwortlichkeiten innerhalb des Unternehmens festlegt. Anderseits werden die Verantwortlichkeiten für die Meilensteine in Meilenstein-Verantwortlichkeitsmatrix für jedes Projekt spezifisch festgelegt.

Tabelle A.5 zeigt die grundlegende Verantwortlichkeitsmatrix für XiTrust. Diese ist eine Ergänzung der in Abschnitt A.3.4 beschriebenen Rollen sowie der Rollenzuordnung in Tabelle A.2. Tabelle A.6 stellt die Bedeutung der Symbole klar.

	Georg Lindsberger	**Product Owner XiTrust**	**Product Owner Kunde**	**Entwicklungs- & Supportteam**
Projektplanung	I	C	P|D	C
Lieferantenmanagement		C	P|D	
Aktivitätsplanung und -durchführung		C	C	P|X
Berichtswesen			P|X	C
Kalkulation	C		P|X	

Tabelle A.5: Grundlegende Verantwortlichkeitsmatrix

Symbol	**Bedeutung**
D	Entscheidet endgültig
d	Entscheiden gemeinsam
X	Arbeitet an der Aufgabe
P	Leitet Arbeit und kontrolliert Fortschritt
C	Muss konsultiert werden
I	Wird informiert
T	Trainer

Tabelle A.6: Bedeutung der Symbole in der Verantwortlichkeitsmatrix

Zusätzlich zur grundlegenden Verantwortlichkeitsmatrix müssen spezifisch für jedes Projekt weitere Verantwortlichkeiten festgelegt werden. Die Projektverantwortlichkeitsmatrix hat das gleiche Format wie die grundlegende Verantwortlichkeitsmatrix, jedoch enthält sie projektspezifische Aufgaben und Rollen. Die Projektverantwortlichkeitsmatrix wird mit Hilfe der Stakeholderanalyse erstellt, die sich auch externe Rollen (Mitarbeiter des Kunden, Lieferanten, etc.) enthält.

Die Verantwortlichkeitsmatrix hilft auch potentielle Engpässe oder Überlastungen zu erkennen. Sind einer Rolle zu viele Symbole zugeordnet ist sie damit eventuell überlastet. Sind einer Aufgabe zu viele Symbole zugeordnet, ist mit einer verlängerten Bearbeitung oder Entscheidungsfindung zu rechnen. In beiden Fällen sollte überlegt werden, ob man die daraus resultierende Projektverlängerung bewusst in Kauf nimmt oder die Verantwortlichkeitsmatrix nochmals ausdünnt.

REFERENZEN: (Andersen et al., 2004, S. 103-112)

Risiko Analyse

Die Risikoanalyse oder auch Unsicherheitsanalyse ist wichtig um herauszufinden, welche Faktoren eine erfolgreiche Fertigstellung des geplanten Projekts verhindern könnten. Dabei sollten Risiken einerseits auf der Projektebene und andererseits auf der Meilensteinebene analysiert werden. Daher wird die Risikoanalyse nach bzw. als Ergänzung der Planung durchgeführt.

Risiken sind vor allem in den folgenden Bereichen zu finden.

- **Technologie**: z.B.: Bekanntheitsgrad, vorhandenes Wissen, bekannte Probleme, Verfügbarkeit
- **Lieferanten**: z.B.: Zuverlässigkeit, Bedeutung für das Projekt, Vertrauen, verwendete Prozesse
- **Ressourcen**: z.B.: eigenes Personal, Personal des Kunden, Hardware, Räume, Kundeninfrastruktur
- **Stakeholder**: z.B.: Meinungsänderungen, Zielkonflikte, Wahrnehmen der eigenen Verantwortung
- **Projektumgebung**: z.B.: Behörden, Umwelt
- **Abhängigkeiten zu anderen Projekten**: z.B.: möglicher später Abschluss eines anderen Projekts, mehrere Projekte beim gleichen Kunden

Weitere Risiken sind in (Andersen et al., 2004, S. 122) aufgelistet.

Für jedes gefundene Risiko wird eine Zeile der Risikomatrix (Tabelle A.7) ausgefüllt. Dies beinhaltet Maßnahmen zur Risikominimierung und Reaktionen auf ein eingetretenes Risiko, sowie die Zuweisung von Verantwortung für die jeweilige Maßnahme.

Es ist nicht notwendig die Risikoanalyse für alle Meilensteine zu Projektbeginn durchzuführen. Dies kann auch rechtzeitig vor oder zum Starttermin des jeweiligen Meilensteins erledigt werden.

Meilenstein	**Risiko**	**Wahrscheinlichkeit**	**Auswirkung**	**Maßnahmen**	**Verantwortlicher**
leer, wenn das ganze Projekt betroffen ist	Art des Risikos	Wahrscheinlichkeit, dass das Risiko auftritt	Größe der Auswirkung auf das Projekt, z.B.: Hoch (Zeit)	Maßnahme zur Risikominimierung und Reaktionen auf ein eingetretenes Risiko	Verantwortlich für die Durchführung der Maßnahme, z.B.: Product Owner Kunde

Tabelle A.7: Risikomatrix

REFERENZEN: (Andersen et al., 2004, S. 120-125, 214-216)

Projektmandat und Projektgesamtübersicht

Alle erarbeiteten Projektinformationen werden zur späteren Referenz und Ergänzung im Projektmandat festgehalten. Dort ist stichwortartig zusätzlich auch noch die Ausgangslage, Projektbeschränkungen sowie das Projekt Budget zu beschreiben. Weiters ist im Projektmandat das Projekt Burndown Chart für die spätere Steuerung des Entwicklungsanteils zu finden.

Zur besseren Übersicht ist das Projekt noch mit seinen wichtigsten Informationen in die Projektgesamtübersicht einzutragen. Für schnellen Zugriff wird in dieser Liste das Projektmandat verlinkt.

Dokumente im SVN:

- Vorlage Projektmandat: `svn:/Vorlagen/Projektmanagement/2009-07-15 Projektmandat.xltx`
- Projektübersicht: `svn:/Kundenprojekte/2009-07 Projekt- und Resourcenübersicht.xlsx`

Projektinfrastruktur

Die Projektinfrastruktur besteht aus den für das Projekt notwendigen Entwicklungs-, Test und Abnahmesystemen. Für manche Projekte wird die vorhandene Infrastruktur ausreichen, für andere muss diese um weitere Komponenten erweitert werden, z.B. wenn neue Technologie zum Einsatz kommt. Die dafür benötigte Zeit muss im Projekt eingeplant werden.

Die Vorbereitung der Infrastruktur findet vor dem Start der Entwicklung statt. Damit ist sichergestellt, dass nicht während des Projekts ein Technologiewechsel stattfin-

den muss, der zusätzliche Kosten und Zeitaufwand verursacht. Weiters werden dabei mögliche Probleme mit einer neuen Technologie aufgedeckt und frühzeitig Gegenmaßnahmen geplant werden.

A.4.3 Projektplanung

Eine realistische Planung ist ein wichtiger Grundstein der erfolgreichen Projektabwicklung. Sind die durch die Planung festgelegten Termine zu aggressiv, wirkt dies demotivierend auf die Mitarbeiter und qualitätsmindernd auf das Produkt. Sind die Termine zu weit in die Zukunft gesetzt, hat sich möglicherweise der Bedarf des Kunden wesentlich verändert.

Die Projektplanung ist keine isolierte Aktivität zu einem bestimmten Zeitpunkt im Projekt. Planungsaktivitäten finden in allen Phasen eines Projekts statt. Die ersten Daten für die Projektplanung stammen bereits aus der Vertriebsphase des Projekts, in der in der Vorkalkulation Aufwände geschätzt werden. Die Grobplanung und erste Detailplanung ist Teil der Projektvorbereitung. Die Vervollständigung des Detailplans ist Teil der Umsetzung und des Abschließens. Diesen Sachverhalt verdeutlich auch Abbildung A.6. Sie zeigt die Planung als phasenübergreifenden Prozess.

Ein wesentliches Merkmal guter Planung ist auch, dass sie zum richtigen Zeitpunkt stattfindet. Findet sie zu früh statt, sind zu wenig oder inkorrekte Daten verfügbar und der Plan muss oft geändert werden. Findet sie zu spät statt, wurden wesentliche Entscheidungen bereits getroffen oder versäumt.

Die folgenden Abschnitte beschreiben die Elemente der Projektplanung und welche Daten dazu notwendig sind.

Der Meilensteinplan

Der erste Schritt in der Projektplanung ist die Erstellung eines Meilensteinplans (Andersen et al., 2004, S. 76-93). Ein Meilenstein beschreibt immer einen Status, niemals eine Aktivität. Er ist ein Kontrollpunkt, der sicherstellen soll, dass das Projekt in die richtige Richtung geht. Der Meilensteinplan zeigt die Abhängigkeiten zwischen den Meilensteinen. Die notwendigen Schritte für das Erstellen sind:

- Lege die Meilensteine als Statusbeschreibung fest. Es muss messbar sein, ob der Status erreicht wurde. Manchmal ist es auch notwendig einem Meilenstein eine Bedingung hinzuzufügen.
- Stelle fest, welche Projektziele parallel erreicht werden können.
- Stelle Abhängigkeiten zwischen den Meilensteinen fest.

Der Meilensteinplan wird immer unabhängig von spezifischen Aktivitäten erstellt. Dies verhindert ein frühzeitiges Einschränken auf genau einen Lösungsweg.

Typischerweise werden Meilensteine bei XiTrust Ziele aus den folgenden Projektbereichen darstellen.

Projektvorbereitung. Tätigkeiten der Projektvorbereitung sind Tätigkeiten, die vor der Erbringung der eigentlichen Entwicklung oder Dienstleistung durchgeführt werden müssen. Dazu gehören typischerweise Tätigkeiten wie der Aufbau der Projektinfrastruktur, Erstellen von Grobarchitektur und -design, Schulung und Wissensaufbau für eine neue Technologie, etc.

Entwicklung. Alle Softwareentwicklungsaktivitäten.

Dienstleistungen. Dienstleistungen in einem Projekt umfassen Tätigkeit wie z.B.: Installationen, Konsultationen, Analysen, Workshops und Schulungen.

Projektabschluss. Für einen erfolgreichen Projektabschluss sind Aktivitäten wie Abnahme mit dem Kunden, Übergabe aller Lieferungen, Abschlussschulungen und Abrechnung des Projekts notwendig.

Abbildung A.9 zeigt ein Beispiel für einen Meilensteinplan. Die logische Verknüpfung (Pfeil) bedeutet, dass der Meilenstein am Ende des Pfeils nicht vor dem Meilenstein am Anfang des Pfeils abgeschlossen werden kann. M2 kann nur abgeschlossen werden, wenn E2 und M1 abgeschlossen sind. Die Arbeit für einen abhängigen Meilenstein (M2) kann jedoch vor Abschluss der anderen Meilensteine (E2 und M1) gestartet werden.

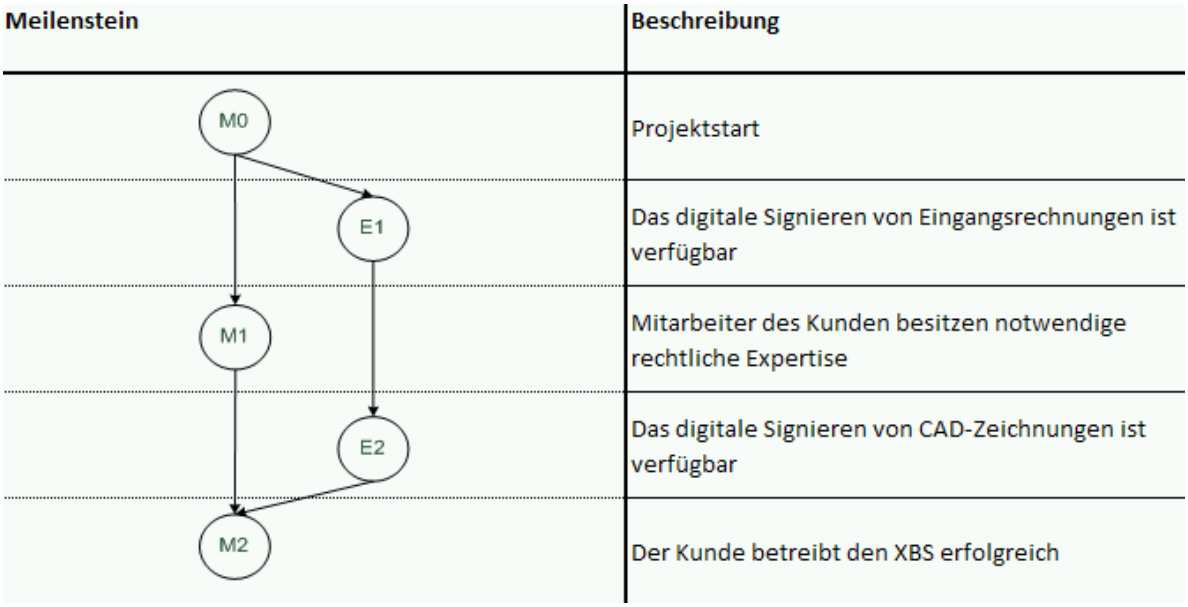

Abbildung A.9: Beispiel für einen Meilensteinplan

Die Meilensteinplanung wird nicht vom Product Owner Kunde allein vorgenommen. Speziell bei hohen Entwicklungsanteilen ist die Meinung des Kunden für eine entsprechende Aufteilung auf verschiedene Releases und damit Meilensteine einzuholen.

Dazu werden für jedes Release die Anforderungen in Pflichtanforderungen und Sollanforderungen unterteilt. Pflichtanforderungen müssen im Release enthalten sein. Sie machen das Release für den Kunden einsetzbar. Sollanforderungen erhöhen den Wert des Releases. Sie können jedoch aus dem Release entfernt werden, sollte sich herausstellen, dass der Arbeitsaufwand unterschätzt wurde. Auch bei Projekten mit nur einem Release sollte nach Pflicht und Sollanforderungen priorisiert werden.

Der Terminplan

Der Terminplan wird auf Basis folgender Daten erstellt:

- Meilensteinplan
- Geschätzte Gesamtsumme der Story Points für den Entwicklungsanteil (aus dem Projekt-Backlog)
- Summe der voraussichtlich benötigten Personentage für Dienstleistungen
- Summe der voraussichtlich benötigten Personentage für Umsetzung der Stakeholderstrategie und Risikominimierungsmaßnahmen
- Vorgesehener Projektzeitraum (z.B. März - Juli 2010)
- Ressourcenverfügbarkeit für den vorgesehen Projektzeitraum
- Verfügbarkeit externer Partner (Kunden, Lieferanten, Behörden, etc.) für den vorgesehen Projektzeitraum
- Vom Kunden bevorzugter Projektendtermin

Im Folgenden ist beschrieben, wo diese Daten herkommen und wie sie verwendet werden.

Der Terminplan wird auf Basis des Meilensteinplans erstellt. Er fügt damit der durch den Meilensteinplan festgelegten Projektstruktur die zeitliche Komponente hinzu. Der Zeitraum zwischen dem ersten und letzen Meilenstein legt damit die Projektdauer fest. Für den Terminplan werden für jeden Meilenstein ein Starttermin und die ressourcenintensivsten Aktivitäten mit deren Gesamtdauer bestimmt.

Der **Starttermin** eines Meilensteins wird vom Projektleiter bestimmt. Dabei sind der Projektstart und eventuelle Abhängigkeiten von logisch vorgereihten Meilensteinen zu berücksichtigen. Der Starttermin muss nicht unmittelbar an den Endtermin des vorrangegangenen Meilenstein anschließen er kann, je nach Ressourcenverfügbarkeit und Abhängigkeiten zwischen den Aktivitäten auch vor oder nach diesem liegen.

Die **Gesamtdauer der Aktivitäten** werden aus den beiden großen Gruppen Entwicklung und Dienstleistung, wie im Folgenden beschrieben, jeweils getrennt berechnet.

Die **Entwicklung** bzw. ein Release wird auf Basis der bisher möglichen Teamgeschwindigkeit, sowie der Verfügbarkeit der Entwicklungsressourcen geplant. Die

bereits verplanten Story Points sind im Projektmandat der anderen zu diesem Zeitraum geplanten Projekte ersichtlich. Die Anzahl der gleichzeitig entwickelten Projekte sollte möglichst gering sein. Die Planung wird wie in Tabelle A.8 durchgeführt.

	Beispiel
Teamgeschwindigkeit	10
- Story Points pro Sprint für Support	2
- Story Points pro Sprint, die in diesem Zeitraum für ein anderes Projekt verplant wurden	4
= **verfügbare Story Points pro Sprint**	**4**
Gesamtsumme Story Points im Release	12
: verfügbare Story Points pro Sprint	4
= **Anzahl der benötigten Sprints**	**3**
x Anzahl der Kalenderwochen in einem Sprint	2
= **Gesamtdauer der Entwicklung in Kalenderwochen**	**6**

Tabelle A.8: Berechnung der Entwicklungsdauer

Dabei ist zu beachten, dass die Gesamtsumme der Story Points nicht nur aus den Pflichtanforderungen errechnet wird. Die Sollanforderungen sind der sogenannte Featurepuffer für den Projekttermin. Ist es nicht möglich einen solchen Featurepuffer zu verwenden, sollte ein Zeitpuffer eingeplant werden. Auch eine Kombination aus beidem ist möglich.

Dienstleistungen werden auf Basis von Personentagen berechnet. Dazu wird zuerst die Dauer der Aktivität im Idealfall, also ohne Unterbrechung geschätzt. Danach wird je nach Aktivität die Ressourcenverfügbarkeit aus der Projektübersicht miteinbezogen und eine Gesamtdauer in Kalenderwochen wie in Tabelle A.9 errechnet. Zusätzlich muss bei Aktivitäten, bei denen externe Partner involviert sind, deren Verfügbarkeit eingeholt und eingeplant werden. Ist der Partner für die Gesamtdauer der Aktivität nicht oder nur eingeschränkt verfügbar, muss die Aktivität entsprechend verlängert werden.

	Beispiel
Dauer der Aktivität	5 PT
: Durchschnittlich verfügbare Kapazität pro Kalenderwoche der notwendigen Person oder Gruppe	2 PT
= **Gesamtdauer der Aktivität in Kalenderwochen**	**3 KW**

Tabelle A.9: Berechnung der Gesamtdauer einer Aktivität

Ist die Dauer berechnet, müssen noch die Abhängigkeiten und möglichen Parallelitäten der Aktivitäten berücksichtigt werden. Die längste nicht parallele Kette von Aktivitäten bestimmt dann die Gesamtdauer (nicht den Gesamtaufwand!) aller Aktivitäten eines Meilensteins.

Tabelle A.10 führt ein Beispiel an, wie der Endtermin eines Meilensteins berechnet werden kann.

	Beispiel
Starttermin des Meilensteins	12.04.2010
+ Max(Gesamtdauer Entwicklung, Gesamtdauer Dienstleistungen)*	Max(6,3) = 6 KW
= **Endtermin des Meilensteins**	24.05.2010

*gilt nur unter der Annahme, dass Entwicklungstätigkeiten und Dienstleistungen hauptsächlich parallel abgewickelt werden können.

Tabelle A.10: Berechnung des Termins für einen Meilenstein

Zusätzlich müssen in den Terminplan noch die Ergebnisse aus der Stakeholder- und Risikoanalyse einfließen. Der Aufwand für die dort geplanten Maßnahmen muss im Terminplan entsprechend berücksichtigt werden. Beispielsweise könnte eine Schulung für das Minimieren eines Technologierisikos das Verschieben des Starts der Entwicklungstätigkeiten notwendig machen. Weiters sollte der Terminplan Zeit für Aktivitäten einkalkulieren, die im Fall des Eintretens eines Risikos notwendig sind.

Für das Gesamtprojekt gibt es daher zwei Enddaten. Das erste und frühere ist das Fertigstellungsziel, das ausdrückt, wann das Projekt bestenfalls fertig ist. Das zweite ist der Fertigstellungstermin, der entsprechende Risikoreaktionen inkludiert. Sind beide identisch, steht der Terminplan unter der Annahme, das kein Risiko eintritt (DeMarco, 2001, S. 209).

A.4.4 Projektumsetzung

Laufende Planung

Die laufende Planung berücksichtigt einerseits die weitere Meilensteinplanung und andererseits die Detailplanung für die nächsten zwei Wochen der Umsetzungsphase. Die erste Detailplanung ist spätestens vor Beginn der ersten Woche der Projektumsetzungsphase vorzunehmen. Die laufende Meilensteinplanung ist spätestens vor Beginn eines neuen Meilensteins fällig.

Bei der Meilensteinplanung wird für der zu beginnende Meilenstein auf Gültigkeit überprüft. Dies betrifft die antizipierten Aktivitäten und Schätzungen, aber auch die Existenz des Meilensteins selbst. Ein geändertes Ziel kann einen Meilenstein

hinfällig machen. Notwendige Änderungen sind in der Planung entsprechend zu berücksichtigen.

Die Detailplanung basiert auf der Wochenplanung und der Sprintplanung. Es ist jedoch zu empfehlen, auch die verschiedenen Aktivitäten der Projektvorbereitung, wie beispielsweise das Aufbauen der Projektinfrastruktur, einer Detailplanung zu unterziehen.

In der **Sprintplanung** werden die Entwicklungsaktivitäten des Projekts geplant. Diese ist hinreichend in Abschnitt A.3.6 beschrieben. Das Priorisieren bei mehreren gleichzeitig laufenden Projekten ist in Abschnitt A.3.8 behandelt.

Die **Wochenplanung** hat die Planung der Dienstleistungen zum Inhalt. Sie besteht aus zwei Teilen, erstens der Abstimmung der Product Owner untereinander und zweitens der Planung gemeinsam mit den notwendigen Mitarbeitern, ähnlich der Sprintplanung.

Die Abstimmung der Product Owner dient dem kurzfristigen Priorisieren der Projekte und der Planung der unmittelbar bevorstehenden Aufgaben. Dazu müssen folgende Daten berücksichtigt werden

- Welchen Fortschritt wurde in den einzelnen Projekten erzielt?
- Welche Ziele in den einzelnen Projekten sollen in dieser Woche erreicht werden?
- Welche Kapazitäten stehen in dieser Woche zur Verfügung?

Projekte mit geringem Fortschritt oder dringenden Termin können so höher priorisiert und die Aufgaben entsprechend der Kapazitäten verteilt werden. Weiters sind eventuell noch zu planende Kundentermine aus neuen Projekten oder Supporttätigkeiten abzustimmen, bevor sie dem Kunden kommuniziert werden.

Anschließend findet die Detailplanung gemeinsam mit den Mitarbeitern statt. Die in der Abstimmung festgelegten Projektziele werden den Mitarbeitern mitgeteilt und gemeinsam entsprechende Tasks formuliert. Die Festlegung des Wochenplans erfolgt auf dem Wochenplanungsboard, dieses ist wie das Sprintboard organisiert. Dabei entsprechen Projektnamen den Stories und die Farbe eines Taskkärtchens zeigt die Zuordnung zu einem Mitarbeiter an.

Das Priorisieren der Tasks erfolgt nach dem Top-Load-Prinzip. Zusätzlich können Tasks zu Gunsten eventuell ungeplanter Ereignisse oder Schätzfehlern als optional nominiert werden. Wie in der Sprintplanung entscheiden die an der Durchführung beteiligten Mitarbeiter, ob sie sich zur geplanten Arbeitsmenge verpflichten.

Zur Fortschrittskontrolle und Feinabstimmung der Wochenplanung findet zur Wochenmitte ein Synchronisationsmeeting, genannt Projekt Illy, aller an der Durchführung der geplanten Aufgaben beteiligten Mitarbeiter statt. Die Inhalte entsprechen denen des Daily Scrum.

Tabelle A.11 zeigt die vorgesehenen Termine für Meetings der Wochenplanung.

Mo	08:30 - 09:00	Product Owner Abstimmung
	09:00 - 09:30	Wochenplanung
Di		
Mi	09:00 - 09:15	Projekt Illy
Do		
Fr		

Tabelle A.11: Wochenplanungskalender

Softwareentwicklung

Die Umsetzung der Softwareentwicklung eines Projekts erfolgt gemeinsam mit anderen Projekten über den in Abschnitt A.3 beschriebenen Prozess. Im Rahmen der Projektabwicklung sind daher bei der Umsetzung der Projektsoftwarentwicklung zwei Komponenten wichtig. Dies sind einerseits das Einflechten des Projekt-Backlogs in die Produktentwicklung und andererseits die Steuerung der Umsetzung zusammen mit dem POX.

Das Projekt-Backlog wird in das Produkt-Backlog in Trac aufgenommen. Dort wird auch vermerkt, welchem Projekt die Stories zuzuordnen sind. Dies dient der einfacheren Kontrolle der noch offenen Aufwände. Vor Beginn der ersten Iteration muss dann eine entsprechende Menge an Stories für die Planungssitzung aufbereitet werden. Für das angegebene Beispiel (Tabelle A.8) wären dies Stories mit einem Gesamtaufwand von etwas mehr als 4 Story Points.

Die Steuerung der Projektsoftwareentwicklung basiert auf dem Projekt Burndown Chart im Projektmandat. Die Trendlinie des Burndown Charts zeigt an, wann zu erwarten ist, dass alle Stories eines Projekts vollständig entwickelt sind. Liegt das erwartete Datum hinter dem geplanten Datum ist das Projekt in der Entwicklungspriorität höher zu stufen. Bei einer erwarteten vorzeitigen Fertigstellung kann die Priorität verringert werden (siehe A.3.8).

Ist erkennbar, dass trotz aller Bemühungen der Releasetermin nicht gehalten werden kann, gibt der POK dies umgehend dem Kunden bekannt. Gemeinsam wird eine Lösungsstrategie erarbeitet. Diese sollte eine Kombination aus dem Verringern des Umfangs, Aufteilen auf mehrere Releases und, als letzter Ausweg, dem Verschieben des Releasetermins sein.

Dienstleistung

Wie die Entwicklung müssen auch die Dienstleistungsaktivitäten gesteuert werden. Dazu sind folgende Schritte notwendig:

- Aktualisieren der bereits verbrauchten Personentage und der Aktivitätenstati

- Aktualisieren der noch benötigten Personentage der entsprechenden Ressourcen
- Kontrollieren, ob mit den vorhandenen Ressourcen die noch offene Arbeitsmenge bis zum nächsten Meilensteintermin bearbeitet werden kann
- Planen eventueller Korrekturmaßnahmen, wie das Verringern des Aufwands oder Hinzuziehen von weiteren Ressourcen

Das Aktualisieren sollte spätestens vor der Abstimmung am Montag stattfinden, da eventuell notwendige Korrekturmaßnahmen mit den anderen Product Ownern abgesprochen werden sollten.

A.4.5 Projektabschluss

Kundenabnahme

Die Kundenabnahme wird meist eine Voraussetzung für das Erreichen des letzten Meilensteins eines Projekts sein. Auf Kundenseite hängt davon meist auch die Freigabe der Zahlung ab.

Bei der Abnahme wird die Software anhand der gestellten Anforderungen validiert. Ist der Kunde mit dem Ergebnis zufrieden, gilt die Software als abgenommen und die Entwicklung ist abgeschlossen. Mit einer agilen Vorgehensweise hat der Kunde die Software schon mehrmals vor der Abnahme begutachtet, daher sollte es dabei zu keinen großen Überraschungen kommen. Dennoch ist es sinnvoll die Abnahme als formalen Schlussstrich für die Entwicklung einzusetzen.

Die Durchführung der Abnahme ist eine Kombination aus einer Live-Demonstration wie beim Sprint Review und einer Reihe von vordefinierten manuellen und/oder automatischen Abnahmetests. Dazu ist es natürlich notwendig die Entwicklung nach den Fertigstellungskriterien für ein Projekt abzuschließen.

Abrechnung

Die Projektabrechnung beinhaltet nicht nur die Rechnungslegung an den Kunden sondern auch die Nachkalkulation des Projekts. Bei der Nachkalkulation werden auf Basis der Daten aus der Projektvorkalkulation Abweichungen festgestellt. Damit lässt sich eine Aussage über die tatsächliche Rentabilität des Projekts treffen. Weiters werden dabei Erfahrungen und Verbesserungsmöglichkeiten für die Vorkalkulation von Folgeprojekten gewonnen.

Folgende Daten fließen in die Nachkalkulation ein.

- Verbrauchte Manntage laut Zeitaufzeichnung inkl. Faktoren für Überstunden
- Anzahl der Sprints für die Entwicklung

- Dem Projekt zurechenbare Kosten für Zulieferungen (Schulungen, Software, Hardware, externe Mitarbeiter, etc.)
- Reisekosten
- Dem Projekt zurechenbare Verwaltungs- und Gemeinkosten

Die Struktur der Nachkalkulation muss der Struktur der Vorkalkulation entsprechen, damit die Werte vergleichbar bleiben. So dürfen dem Projekt bei der Nachkalkulation auf keinen Fall andere Gemeinkosten zugerechnet werden, als bei der Vorkalkulation.

Projekt Retrospektive

Die Projekt Retrospektive betrachtet den gesamten Zeitraum der Konzepterstellung bis zum Zahlungseingang. Daher findet sie auch nach Letzterem statt. Sie ist genauso strukturiert wie eine Sprint Retrospektive und ist eine Ergänzung zu den unternehmensweit stattfindenden Verbesserungsanstrengungen. Dabei werden Themen behandelt, die die Projektabwicklung betreffen. Dazu gehören vor allem:

- Kommunikation und Zusammenarbeit mit dem Kunden
- Anforderungsanalyse zusammen mit dem Kunden
- Formen der Projektbasis
- Projektplanung
- Projektabschluss
- Budget und Finanzen

Es sind aber auch andere Themen vorstellbar, solange sie in den Bereich der Projektabwicklung fallen. Wie auch bei der Sprint Retrospektive sollte vor Beginn der Projekt Retrospektive die im eben abgeschlossenen Projekt größte Herausforderung als Bearbeitungsthema ausgewählt werden. Dies kann und soll natürlich auch bedeuten, zu analysieren, welche Aspekte der Herausforderung besonders gut bewältigt wurden. Weiters können mehrere Projekte mit ähnlich gelagerten Abläufen gemeinsam analysiert werden.

TEILNEHMER: POK, POX, SM, Team, Kundenrepräsentanten

DAUER: max. 4h

Vorbereitung, Ablauf und Ergebnisse sind der Beschreibung der Sprint Retrospektive (siehe A.3.6) zu entnehmen.

Abschlussfeier

Oft sind an den Abschlusstätigkeiten eines Projektes nur noch wenige oder ein Mitarbeiter beteiligt. Die Projektabschlussfeier setzt damit einen gemeinsamen Schlus-

spunkt für alle die Zeit und Arbeit in das Projekt investiert haben. Sie dient der Anerkennung der Leistungen aller Beteiligten und setzt somit einen positiven Schlusspunkt.

Dies ist insbesondere bei Projekten wichtig, die nur unter besonderem Einsatz der Mitarbeiter zu einem positiven Abschluss gebracht werden konnten. Weiters werden dabei wichtige Erfahrungen unter den Mitarbeiter ausgetauscht, indem informell über prägende Erlebnisse im Projekt geplaudert wird.

Die Größe der Abschlussfeier ist natürlich abhängig von der Größe des abgeschlossenen Projektes. Nach kleineren Projekten wird ein Zusammensitzen bei Kaffee und Kuchen ausreichen, in anderen Fällen ist auch ein gemeinsames Abendessen beim Lieblingsitaliener vorstellbar.

A.5 Wartung und Support

Der Bereich Wartung und Support definiert Prozesse für die Themen Supportleistung und -dokumentation, Supportverträge und Softwarekorrekturen. Diese Prozesse sind Teil des Transformationsprojekts, werden aber nicht vom Autor dieser Arbeit definiert.

Einige besonders zu berücksichtigende Punkte bei der Definition von Prozessen für Softwarekorrekturen sind dennoch im Folgenden aufgeführt.

- Priorisieren der Softwarekorrekturen inkl. rechtzeitiges und begründetes Ablehnen von Korrekturwünschen
- Reaktions- und durchlaufzeiten für die priorisierten Korrekturen
- Branchen und Rückintegrieren des Korrekturcodes in Bezug auf die Hauptcodebasis des betroffenen Produkts
- Installation des Korrekturcodes beim Kunden

Die Entwicklung der Softwarekorrekturen sollte möglichst vom dem Team, das den betroffenen Code entwickelt hat, durchgeführt werden. Dieses erhält dadurch wichtiges Feedback über Codequalität und Wartbarkeit des Codes. Weiters ist es zusätzliche Motivation, den Code möglichst fehlerfrei zu halten, da neue Herausforderungen in Projekten interessanter sind als die Fehlersuche.

A.6 Produkt-Innovation

Der Bereich Produkt-Innovation soll in Zusammenarbeit von dem Kunden XiTrust und allen interessierten XiTrust-Mitarbeitern neue, innovative Ideen für die Erweiterung der XiTrust-Produktlinie hervorbringen. Die genaue Definition der Prozesse für diesen Bereich sind im Transformationsprojekt zeitlich nach Ende dieser Arbeit angesetzt. Daher werden an diesem Punkt nur Vorschläge für die Umsetzung gemacht.

Sammeln von Ideen

Haben Mitarbeiter Ideen für die Erweiterung der Produktline sollten diese an einer Stelle gesammelt werden. Dazu bietet sich das Tool Trac an, in dem auch die Anforderungen verwaltet werden. Dort kann eine eigene Kategorie Ideen eingerichtet werden.

Für das Aufschreiben von Ideen ist ein bestimmtes Schema vorgegeben, durch dessen Ausfüllen das Weiterbewerten einer Idee erleichtert. Tabelle A.12 zeigt einen Vorschlag für ein solches Schema. Das Ausfüllen kann in Zusammenarbeit mit einem Innovations-Verantwortlichen erfolgen, der den jeweiligen Mitarbeiter bei der Formulierung seiner Idee unterstützt. Es ist jedoch darauf zu achten, dass sich der Aufwand für die Beschreibung einer Idee in Grenzen hält, da viele Ideen nicht in die Entwicklung übernommen werden können.

Kurzbeschreibung*	kurzer, prägnanter Name der Idee
Betroffenes Produkt*	z.B. e3Server
Betroffene Komponenten*	z.B. Workflow-Engine, Signatur-Kommandos
geschätzter Aufwand in Story Points	z.B. 40
mögliche Kunden	wenn bereits bekannt
Beschreibung*	längere ausführlichere Beschreibung
Auslöser der Idee*	z.B. Gespräch mit Kunden, Lesen eines Artikels, etc.

* Pflichtfelder

Tabelle A.12: Schema für das Beschreiben einer Idee (Vorschlag)

Innovations-Workshop

Zweck des Innovations-Workshops ist, einige konkrete, vielversprechende Produktideen zu erhalten, die dann in die Softwareentwicklung eingeplant werden. Dieser soll regelmäßig, z.B. einmal jedes Quartal, stattfinden und kann von einem externen Experten oder dem SM moderiert werden. Im Folgenden ist ein Vorschlag für den Ablauf eines solchen Workshops angeführt.

TEILNEHMER: Kunde XiTrust, POX, interessierte Mitarbeiter, externe Spezialisten (technisch, marktorientiert, Kundenrepräsentanten), Moderator

DAUER: 4h

VORBEREITUNG:

- Bereite Struktur und Aktivitäten für den Workshop vor — Moderator
- Bestimme ein Thema (z.B. Ausbau der Workflow-Engine, Signature-Features, SuperSonic) für den Workshop — Kunde XiTrust
- Filtere die Liste der vorhandenen Ideen anhand des vorgegebenen Themas — Ìnnovations-Verantwortlicher

ABLAUF

- Stelle das Thema und den Ablauf vor — Moderator
- Sammle erste Ideen (z.B. mittels Brainstorming, Brainwalking, etc.) — Alle
- Vorstellen der Ideen auf der Liste — Innovations-Verantwortlicher
- Sammle weitere Ideen basierend auf den vorgestellten Ideen (z.B. mittels Brainstorming, Brainwalking, etc.) — Alle
- Wähle aus allen gesammelten Ideen inkl. der Liste die Top 10 Ideen mittels einer geeigneten Filtertechnik aus — Alle
- Beschreibe neue Ideen anhand des Schemas für die Beschreibung von Ideen (A.12) — Alle
- Finde Möglichkeiten für die Umsetzung der gewählten Ideen — Alle
- Schätze den Aufwand für die Umsetzung der Ideen — Alle

Noch während des Workshops oder auch nach Ende sollten der POX und der Kunde XiTrust eine ROI oder andere Kosten-Nutzen-Rechnung für die gewählten 10 Ideen durchführen. Basierend auf den Ergebnissen dieser Rechnung und den verfügbaren Ressourcen in der Software-Entwicklung sollten die Top 3-5 Ideen in die Softwareentwicklung eingeplant werden.

Abschließen eines Innovations-Zyklus

Geht man von einem Innovationszyklus von einem Quartal aus, ist es nicht zielführend, die in einem Workshop bereits behandelten und aussortierten Ideen weiter in der Ideen-Liste aufzubewahren. Mit dem Fortschreiten der Zeit und damit Veränderungen am Markt werden die Ideen obsolet und fordern daher unnötig Ressourcen, die für eine erneute Bewertung notwendig wären.

Es ist zukunftsorientierter neue Ideen oder Chancen je nach Marktsituation zu generieren, als an alten Ideen festzuhalten. Ähnliches gilt für Ideen, die sich bereits einen längeren Zeitraum unbehandelt in der Ideen-Liste befinden. Daher sollte der Innovationszyklus immer mit dem Konsolidieren und damit Verkürzen der Ideen-Liste abgeschlossen werden.

Zeit für Experimente

Viele Geistesblitze werden erst zu hervorragenden Ideen, wenn es möglich ist, damit zu experimentieren. Um den Mitarbeitern Zeit dafür zu geben, haben Unternehmen unterschiedliche Ansätze gefunden. Google erwartet von seinen Mitarbeitern, 20% ihrer Arbeitszeit mit der Ausarbeitung von eigenen Ideen zu verbringen (Poppendieck

and Poppendieck, 2007, S. 45). Ein anderer Ansatz laut (Kniberg, 2007, S. 74) ist, zwischen dem Ende des einen und dem Anfang des anderen Sprints einen sogenannten *Lab Day* anzuberaumen.

Der Ansatz von Rally (Poppendieck and Poppendieck, 2007, S. 151-152) erscheint dem Autor jedoch am empfehlenswertesten. Nach jedem (Major-)Release haben die Mitarbeiter eine Woche Zeit sich eigenen Experimenten zu widmen. Finden Kunden Fehler nach einem Release sind die Experimente sofort zu stoppen und erst wieder fortzusetzen, wenn der Fehler behoben ist. Damit ist einerseits die sofortige Behandlung von Fehlern gewährleistet und andererseits zusätzlich die Motivation gegeben, das Release fehlerfrei zu halten.

Welcher Ansatz auch gewählt wird, er bietet dem Unternehmen wesentliche Vorteile. Die Mitarbeiter sind motiviert, gut ausgebildet und halten sich selbst technologisch am neuesten Stand. Das Produkt wird verbessert in dem mit der Architektur oder anderen Teilen der Software experimentiert wird. Weiters entstehen kreative, innovative Produktideen. Nicht zuletzt werden in diesem Zeitraum auch systemverbessernde Ideen und Mechanismen gefunden, wie z.B. neue Techniken des Priorisierens, bessere Testframeworks und anderes.