Doctoral Thesis

Review of Agile Software Development Methods in Practice

Ing. Dipl.-Ing. Christian Schindler

Institute for Software Technology
Graz University of Technology

1st Assessor: Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Slany
2nd Assessor: Univ.-Prof. Dipl.-Ing. Dr. Thomas Grechenig

Advisor: Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Slany

Graz, February 2010

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.

[illegible]

Graz, February 2010 Ing. Dipl.-Ing. Christian Schindler

Kurzfassung

Agile Softwareentwicklungsmethoden, besonders Extreme Programming und Scrum, sind seit den Neunzigerjahren sehr populär geworden. Diese Arbeit befasst sich mit dem Einsatz solcher Methoden in der Praxis. Drei Fälle des Einsatzes von agilen Methoden werden beleuchtet. Zuerst wurde ein kleines universitäres Projekt, in dem die Extreme Programming Methodologie eingesetzt wurde, analysiert. Zweitens wurde eine Umfrage über den Einsatz von agilen Softwareentwicklungsmethoden in österreichischen Informationstechnologie Unternehmen durchgeführt. Zuletzt wurde eine Softwareentwicklungsabteilung eines österreichischen Technologie Consulting Unternehmens beim Wechsel von einem wasserfallartigen Entwicklungsprozess zum Scrum Framework beobachtet. Probleme beim Einsatz von agilen Methoden werden aufgezeigt und Lösungsmöglichkeiten vorgeschlagen.

Abstract

Agile software development methodologies and especially Extreme Programming and Scrum have become very popular in the last two decades. This thesis is concerned with the application of agile software development methods in practice. They are reviewed in three aspects. First, an academic project, where the Extreme Programming methodology was applied, is analyzed. Second, the results of a survey about the practical use of agile software development methods in the Austrian IT industry are discussed. Finally the transition of a software development department of an Austrian technology consulting company from a mere waterfall like software development process to the Scrum software development framework was monitored, documented and is analyzed. Emerging problems in the application of agile methods are identified and possible solutions are suggested.

Acknowledgment

I would like to express my deep and sincere gratitude to my advisor, Professor Dr. Wolfgang Slany for his encouraging comments, endless patience, and help in all aspects. I am also indebted to Professor Dr. Thomas Grechenig who agreed to serve on my examining committee on a very short notice.

I am grateful for my colleagues Arndt, Bibiane, Karl and Bruce for the interesting and inspiring discussions as well as the Scrum team "/dev/null", I was allowed to work with over the past 15 months.

Finally, I am forever indebted to my parents, my whole family and friends. I owe my deepest gratitude to my life partner and sunshine Viktoria for her understanding and encouragement when it was most required.

Graz, February 2010 Christian Schindler

Common sense is the most fairly distributed thing in the world, for each one thinks he is so well-endowed with it that even those who are hardest to satisfy in all other matters are not in the habit of desiring more of it than they already have.
(René Descartes 1596-1650)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis agile software development methods are reviewed in three aspects. First, an academic project which was carried out in cooperation with an Austrian software company for Web2.0 applications is analyzed. Second, a survey about the practical use of agile software development methods in the Austrian IT industry was conducted and discussed. Finally the transition of a software development department of an Austrian technology consulting company from a mere waterfall like software development process to the Scrum software development framework was accompanied, documented and analyzed.

1.1 Motivation and Outline

The primary goal of this thesis was to investigate whether the positive attitude towards agile software development methods described in literature [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18] is observable in practice. A software development process alone does not determine whether a project goes well or becomes a failure however. It mainly relies on the involved people, although the process can support or hinder their work depending on its characteristics and how the process is implemented and lived. According to the agile manifest [1], agile methods are strongly human oriented. This is also discussed in the analysis of the htmlButler project in Chapter 3, where an academic project was conducted together with a commercial partner from the software industry. The most interesting discussion concerning this project is about the problems that arose between the developers themselves, the academic project leader and the project management of the cooperating company. One can see that agile methods, although human centered, are not the key factor for a successful project but that *"Everything really interesting that happens in software projects eventually comes down to people"* [19]. Prior to this, the author happened to have experienced a full-fledged heavy weight development process in an Austrian technology company where safety critical hard- and software for public transport and air traffic control communication systems were produced. How this process was really lived was totally different than it was described on paper. In reality it was merely an ad hoc development method with a subsequent heavy test and fix effort to meet the always changing requirements and to reach the quality assurance level so that the product was ready for shipping. That these software projects were never in time nor in budget was no surprise, but the company did not change its process. Agile and light-weight software development methods are described as making it possible to easily react to changes (see [20] and Chapter 2). In the last decade the software industry put the focus on an improved (shortened) time-to-market and this most often by skipping steps of the development process. Changing requirements make it a complicated task to keep the time-to-market down. It was therefore interesting how the Austrian IT-industry copes with this tendency and what software development processes, if any, are used and if they are agile. Out of this motivation, a survey concerning the use of agile software development methods in the Austrian IT-industry was conducted. This survey is described, analyzed and discussed in Chapter 4. As a third part of this thesis, the transition of a software development department within a technology consulting company from a waterfall-like software development process to the Scrum framework was accompanied. Not only the transition but also the practical adoption of the Scrum framework and the problems between the different Scrum roles were monitored over the whole year 2009 and are analyzed in Chapter 5.

1.2 Contributions of this Thesis

Chapter 2 is a foundational chapter where the basics of software development are reviewed. The core activities shared by all software development methodologies, namely specification, design, implementation, validation, verification and maintenance, as well as common software development methods are described. Agile software development methods are described in more detail to set the context for the rest of the thesis.

In Chapter 3, the software development process within an academic project in cooperation with a commercial Web2.0 software company is reviewed. First, the background motivation, the architecture, the way of implementation and a typical use case are described. Then, the problems that emerged in this project during the use of agile practices are reviewed. Finally feedback of the co-working developers was collected and analyzed. The chapter concludes with a summary of the problems, the lessons learned and possible solutions for future projects with similar parameters. In Appendix B, the questionnaire for the developer feedback is described. The following publications about the htmlButler project are incorporated into Chapter 3:

- Pranjal Arya, **Christian Schindler** and Wolfgang Slany. Human-Agent interaction in the light of ontology sharing and large scale cooperation. In *Proc. Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2005)*, pages 158–161, Vienna, Austria, November 28-30 2005. IEEE Computer Society.

- Karl Neuhold, **Christian Schindler** and Wolfgang Slany. The htmlButler Approach: Through Shared Ontologies and Large Scale Cooperation to Enhanced Wrapper Usability. In *Proc. 6th International Conference on Knowledge Management (I-KNOW06)*, pages 35–38, Graz, Austria, September 6-8 2006.

In Chapter 4, the survey of agile software development methods in the Austrian IT-industry is described. From a set of over 400 software companies a sample of 100 was chosen and contacted. 42% of the contacted companies agreed to participate in a telephone interview. The telephone interviews were conducted as structured interviews. The prepared questions are given in Appendix C. The questions concerned a.) current application of agile methods and practices, b.) future plans for the adoption of agile methods and practices and c.) the perception to adherence to delivery dates. To the best of the author's knowledge, this survey is the first and only survey in Austria that targets specifically the use of agile methods. The author has conducted this survey alone, and the outcome was consolidated and already published in the following paper:

- **Christian Schindler**. Agile Software Development Methods and Practices in Austrian IT-Industry: Results of an Empirical Study. In Masoud Mohammadian, editor *Proc. 2008 International Conference on Computational Intelligence for Modelling, Control & Automation (CIMCA 2008), Intelligent Agents, Web Technologies & Internet Commerce (IAWTIC 2008), Innovation in Software Engineering (ISE 2008)*, pages 321–326, Vienna, Austria, December 10-12 2008. IEEE Computer Society.

In Chapter 5, the transition of a technology consulting company's software development department from a mere waterfall like development process to the Scrum framework was accompanied, documented and analyzed. One team's sprint data was collected over the whole year 2009 and acts as a substitute for all other Scrum-teams in the department since the performance and the problems were similar in the other teams. The transition, the performance of the team as well as the problems which arose in the department through the switch to Scrum are discussed. This chapter concludes with a survey among the Scrum-teams about the transition and the situation in the department and a summary about the critical problems and possible solutions. The 23 sprints of 2009 and the related burn-down charts, along some statistical data are collected in Appendix E. The 121 Scrum evaluation survey questions are collected in Appendix F.

Chapter 6 concludes the work of this thesis and sets the experiences of Chapter 3, 4 and 5 into context. Furthermore the problem similarities and possible solutions for the use of agile software methodologies in practice are discussed.

Chapter 2

Software Development

Software (SW) development methodologies systematize the process of developing software. While SW development methodologies differ in details and in arrangement of specific activities, most of them consider the similar core activities. In this chapter, four core activities, namely specification, design, validation/verification and maintenance/evolution are described. Then some well-known (agile) SW development methods are briefly described, since this thesis is concerned with agile development methods in practice.

2.1 Specification

The goal of the specification phase is to set out the functionality that the system which is under development should have. The process of developing a specification is also called requirements engineering. It establishes the service the system should provide and the constraints under which it must operate [21].The outcome of the requirements engineering process is a document containing the functional and non-functional requirements. A functional requirement describes a system service or function whereas a non-functional requirement represents a constraint placed on the system (response time) or on the development process (specific language). The requirements engineering process (see Figure 2.1) consists of four main stages.

Feasibility Study In the feasibility study an estimation is done whether the current technology (hard- and software) is sufficient to develop the system according the identified user needs and whether the budget is adequate. This study is to be done early in the overall software development process and should be done in a quick and cheap way. The out-coming feasibility report should make clear whether further development makes sense and support the decision to go ahead with more detailed work.

Requirements Analysis In this stage system models and prototypes can be developed to help understanding the system requirements and the system to be specified. The requirements are refined through observations of existing systems, discussions with the customer

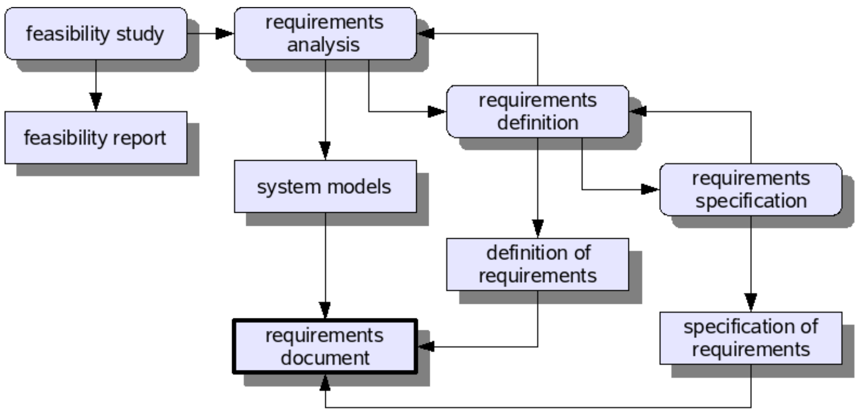


Figure 2.1: A generic requirements engineering process.

and the potential users and observation of the proposed installation sites and so on. The emerging system models become part of the requirements document. The requirements analysis does not stop with the first iteration. New requirements are discovered during the definition and specification process and are fed back into the requirements engineering cycle.

Requirements Definition In the definition stage the gathered information from the analysis is transformed into a document which accurately defines the set of requirements the customer wants. The out-coming document must be written in a way the end-user and customer can understand. New arising analysis or definition issues or contradictions are fed back and influence the whole process.

Requirements Specification In this stage the detailed and accurate descriptions of the system requirements are set out and result in the specification requirements which act as a base for a contract between customer and developer. This document is created in parallel with high level design and therefore cannot be done in one iteration since it is very likely that errors in the requirements definition and contradictions with the design arise which must be corrected and influence the whole specification, definition and analysis cycle.

2.2 Design & Implementation

While design and implementation are often described as separate phases, here they are described together since especially in agile software development processes these two activities are tightly interwoven. The implementation phase is the manifestation of the design in code by means of different programming languages, tools and techniques leading to an executable and testable system. Occurring problems during implementation

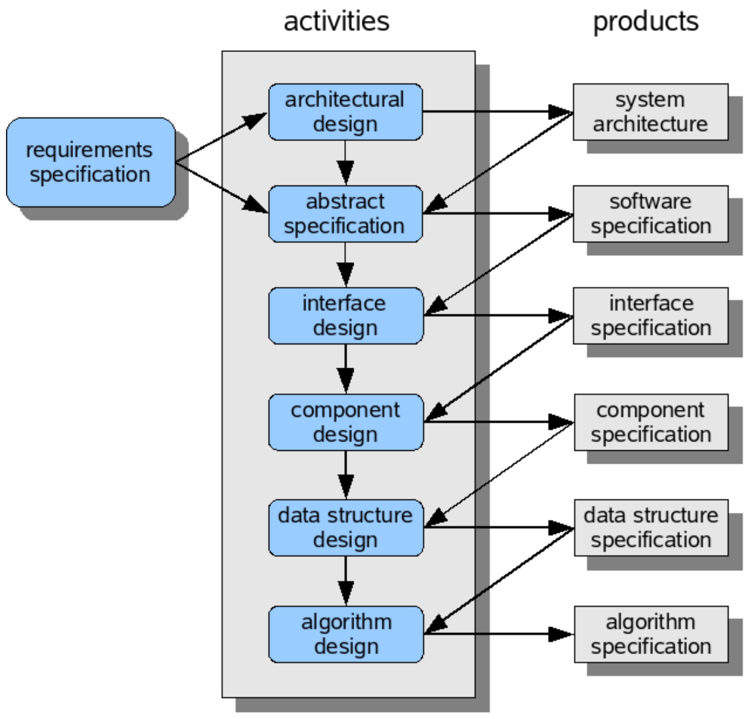


Figure 2.2: General model of a design process.

loop back and have direct influence on the previously made design no matter which process model is used. In the design phase the general structure of the software system is described on different levels of abstraction. Any design problem can be approached in three stages:

Problem Study and Understanding The question that shall be answered in this stage is: “What is the exact problem that has to be solved?” It is often important to “think out of the box”, to be creative. As Albert Einstein put it *“No Problem can be solved from the same consciousness that created it.”* Therefore the problem should be studied from various viewpoints to get different insights into the design requirements.

Identification of Features It is an advantage to create multiple solutions, identify the general features and evaluate them according the simplicity, the degree of reusability and combinability. Multiple possible solutions lead to more features which can be combined and increase the potential of a better design.

Design Description A high-level informal design description makes it easier to discover potential design-flaws and omissions at an early stage. After analysis of this high-level design description the actual design document should be prepared.

The general model of software design as depicted in Figure 2.2 is divided into design activities for the different levels of abstraction and into the emerging abstract formal specification design products. Although the stages are depicted sequentially for ease of understanding, they are practically carried out in parallel with smooth transitions between them. At the beginning of the general design process one starts with an informal design. Within each step the design is refined and becomes more formal, consistent and complete. The gained insights and clarifications of the requirements and the specifications of the functions of the system in each level are fed back to earlier stages. Discovered inconsistencies and errors have an influence on all design activities and henceforth out-coming design documents. The design activities and their emerging specifications are:

Architectural design is leading to the system architecture.
The parts assembling the system and their relationships are identified and documented.

Abstract specification is leading to the software specification.
Each part, their provided services and constraints they have to observe are abstractly specified.

Interface design is leading to the interface specification.
The part's interface to the other involved parts of the system is designed an unambiguously specified. This can be done using formal, algebraic or model-based specification methods.

Component design is leading to the component specification.
A mapping of services to components and their interfaces are designed and documented.

Data structure design is leading to the data structure specification.
A detailed design of the system implementation's data structures is specified.

Algorithm design is leading to the algorithm specification.
The algorithms used in this system which provide services are specified in detail.

2.3 Validation, Verification

The term validation is defined as the process where the tester checks if the (sub)-system under test fulfills its specified task and therefore is for its intended use [22]. Verification on the other hand is focused on distinct phases of development and shall prove the correctness and completeness of emerging artifact of this phase according to its direct specification [22]. Shortly summarized the difference can be expressed by two questions, a.) *validation: "Are we building the right product?"*, and b.) *verification: Are we building the product right?* [23]. In practice both aspects, validation and verification are part of all tests whereas the amount of validation increases with the level of testing. The validation

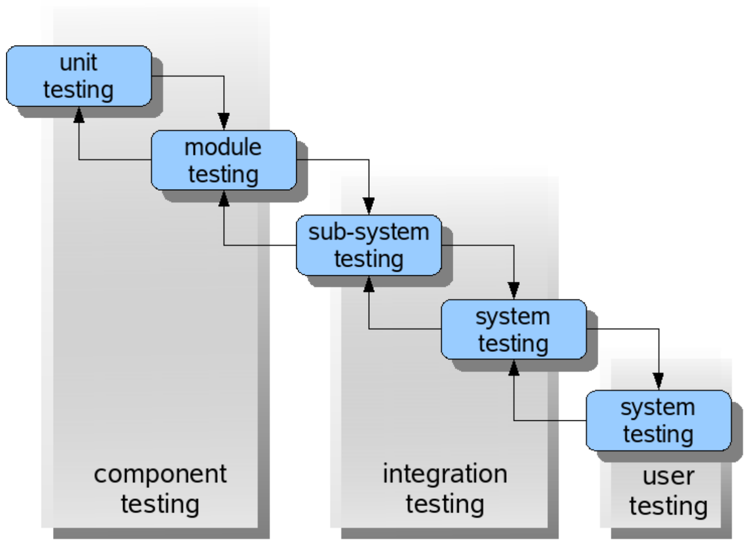


Figure 2.3: A testing process.

and verification process contains static and dynamic techniques. Static techniques can be used at all stages of the software process and are concerned with requirements documents, design diagrams and the source code (metrics). Dynamic techniques can only be used on prototypes and executable (sub) parts of the system. Verification is divided into a.) statistical testing, which is used to test for performance and reliability and b.) defect testing which is intended to find areas where the program diverges from its specification. Systems are built step by step and therefore testing should be carried out in conjunction with system implementation. In Figure 2.3 a very common testing process is depicted which corresponds to the incremental evolution of the software system under development. The steps are sequential but of course not necessarily only unidirectional. This means if at any stage a defect was discovered previous testing stages may be repeated. The stages in the testing process are:

Unit testing The functionality of individual components (methods or procedures) are tested independently without the context of other components.

Module testing The functionality of dependent and coupled components are tested, e.g., classes, packages and abstract data-types (ADTs). Modules are components which are related but not necessarily interdependent. Therefore, they can be testes without other system modules.

Sub system testing In the next higher abstraction layer are subsystems which consists of the former tested modules and build distinct parts of the system (logically related groups of modules) which communicate via interfaces. These interfaces pose a great risk for errors and misinterpretations in system integration and hence must be carefully tested in this stage.

System testing The complete system as it is made up and integrated from the different sub systems is tested for errors in the interactions between the sub system and system components. Also the system is validated if the functional and non-functional requirements are met.

Acceptance testing The final stage of the testing process where the system is confronted with real life data instead of simulated test data. The acceptance test takes place before the system is accepted for operational use. Through the use of “real” data and the testing through customer and intended users, flaws and errors in the system requirements are discovered in terms of unacceptable performance or user’s needs.

After acceptance testing (which is also called alpha testing), a beta-testing phase can be applied where the system is shipped to the customer and/or selected user group to discover errors and requirement irregularities through real-life application.

2.4 Maintenance (Evolution)

When software is needed it can be obtained in different ways (external source, developed in-house, etc.). However the supplier delivers the developed software according to the specified (user) requirements. The maintenance phase of the software begins after the delivery. Maintenance must be thoroughly planned before shipping because afterwards it is deployed outside the development area and at many sites, and therefore making changes after shipping is impossible without a good maintenance/update procedure. There are many potential reasons why the software must be modified: For instance, changes in the environment or hidden defects which must be adapted and corrected by the supplier. Software has an evolutionary nature and without maintenance, software would cease to be useful over time [24] (see Appendix A). Software maintenance is concerned with the modification of the system after completion and shipment to the customer and after it was put into operational use. The term software maintenance is defined differently by different organizations. Some widely agreed definitions are:

“Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment” [25].

“Software maintenance is the totality of activities required to provide cost-effective support to a software system” [26].

“Software maintenance is the modification to code and associated documentation due to a problem or the need for improvement. The objective is to modify the existing software product while preserving its integrity.” [27].

According to [28], all necessary activities of software maintenance can be roughly divided into three categories a) *corrective maintenance* where reported errors are fixed, b) *adaptive maintenance* which is concerned with porting the system to other platforms or to new environments and c) *perfective maintenance* which is concerned with change requests due to new functional or non functional requirements. The distribution between these categories varies between 17%-20% for corrective, 18%-25% for adaptive and 55%-65% for perfective maintenance.

The software maintenance standard ISO/IEC 14764 [29] enhanced the basic three categories by a fourth category namely the *preventive maintenance* which is concerned with modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults [29]. In Table 2.1 the categories of the software maintenance as described in SWEBOK (Software Engineering Body of Knowledge) [26] are depicted. In the IEEE Standard for Software Maintenance [25] a different fourth category was introduced namely the *emergency maintenance* which is concerned with change requests which have to be processed immediately and with highest priority due to the negative influence on the availability of the system or the system's core functionality The IEEE-1219 view of the maintenance categories are grouped by type of schedule (see Table 2.2).

	Correction	**Enhancement**
Reactive	Corrective	Adaptive
Proactive	Preventive	Perfective

Table 2.1: SWEBOK categories of software maintenance.

	Unscheduled	**Scheduled**
Reactive	Emergency	Corrective, Adaptive
Proactive		Perfective

Table 2.2: IEEE categories of software maintenance.

Maintenance costs & cost estimation Maintenance costs vary significantly according the application domain and on the kind of error. For typical business applications maintenance approximately the costs of the system development [30, 31] but can grow much higher for, e.g., embedded- or high-performance & -reliability systems. They vary according to different empirical studies [32] (undertaken from 1969 to 2000) between 50%

and 90%. Maintenance costs are related to many different factors which can be subsumed in two categories, the technical and non-technical factors [33, 21]. Almost half of the maintenance effort is dedicated to involuntary, the other half to discretionary activities [28]. According to a well-known cost model (COCOMO) [34], the estimated annual maintenance effort (AME) is proportional to the initial development effort (DE) and the annual change traffic (ACT).An overview about different maintenance cost estimation approaches can be found in [35]).

Maintenance activities are of corrective, adaptive, perfective and preventive nature. In principle, these activities can be distinguished but in practice they are intertwined. Unrealistically small programs will widely be used to demonstrate maintenance principles, but real maintenance is done on large systems which are actively being used in real world scenarios. Lehman's laws[1] [24] (see Appendix A) describe the principles common to all large, live software systems. In an ideal world software would be build not limited by time and cost boundaries and the system would always be perfect. Since this clearly does not happen, one goes for an approximation, and builds SW systems limited by time and cost and goes for maintenance if it is discovered later that the system is defective or some requirements have changed and therefore the system has to be enhanced.

2.5 Common Software Development Models

Every process has a set of steps, at each step a well-defined task is performed leading to the process goals. At the end of each step the output which must be a clearly defined entity is to be verified according the process definitions. To keep a process feasible it must not have too many steps and a clear objective. Each step must have an entry and an exit criterion, the outcome is just an interim artifact not the final output of the process which are usually documents (specification or design documents or prototypes) and additional information for the management of the process which can take actions to keep the process under control [36] (see Figure 2.4). Since a software process is to be used by more than one project, it must have some general characteristics beside that one to satisfy the project goal. A software process should be predictable, support change, testability and maintainability, early error detection & removal and last but not least support process feedback and improvement. Predictability is necessary to apply the process to different projects. If a process has random outcome it is useless therefore it can be considered a fundamental feature of a process. Predictability concerns time and quality, and hence costs of a process. A process must be able to cope with change since projects are never static and rigid. A process should be prepared for changing requirements, knowledge through user/customer feedback, hardware and human resources. This is especially true if a project lasts for a certain amount of time. Testability & maintainability are also very important features of a process. As stated in previous sections maintenance costs can

[1]Lehman's laws about intrinsic characteristics of software which were formulated over 20 years of studying the software process concerning continuing change, increasing complexity, continuing growth, declining quality, etc.

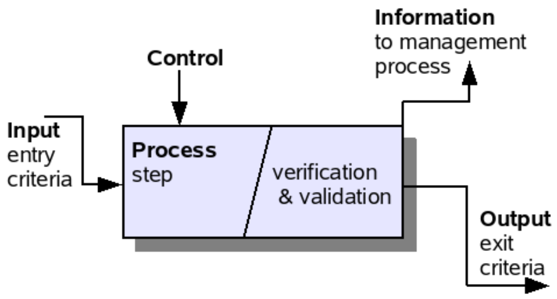


Figure 2.4: Specification of a process step.

generally exceed development costs and therefore reducing the maintenance costs must be an important goal of a process. This can be done through reducing errors in design and coding. Therefore the process must support testability and foster easy modification. Early error detection & removal is vital for a process since errors are made during all phases of development, although the costs for fixing an error differ according the detection in the different phases of the process (see Figure 2.5 [34]).

A process is not a static and rigid step by step sequence. Since the quality, the outcome and hence the costs of a project are largely determined by the process it is important to reduce costs and improve quality. Therefore a software process should be designed as a closed loop. The process must be improved by previous insights from earlier projects and during the process gained knowledge and experience from the distinct phases must be used to improve the rest of the project. This is especially true for iterative processes where feedback from earlier iterations is used as input for the next one. It is not the purpose of this section to list and compare all existing software development models not only because they have been widely documented but rather prepare the general context for the upcoming discussions about agile methodologies in practice.

2.5.1 Build and Fix Model

The build and fix model (see Figure 2.6) is considered to be the least appropriate model [37] for software development since it is totally reactive. However, according to the authors' experience it is today still widely used. The software is produced without a specification and without a proper design therefore the process is based on interaction with the end user which can lead to misunderstandings. This leads to a non predictable project including the time to be finished as well as the real costs of the product. Although most projects are said to be performed using other methods, in practice it often happens that these processes degrade to the build and fix model which often leads to frustration of programmers, customer, and eventually project cancellation. The only advantage of this model is the simplicity which will pay back after the first delivery of the product. The disadvantages are, a.) *no specification*, only oral transmission of the customer's wishes and needs.

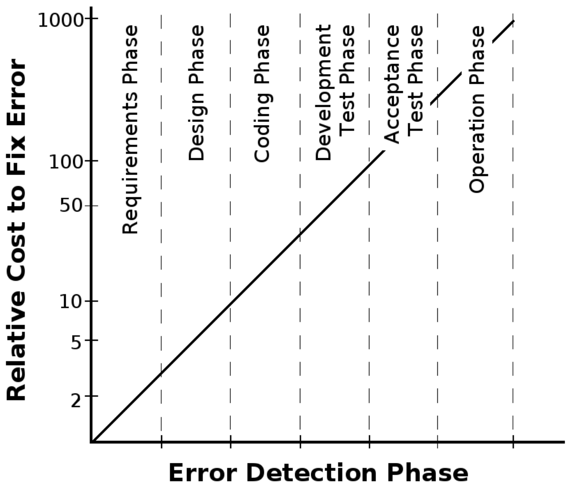


Figure 2.5: Relative costs fixing errors according development phase.

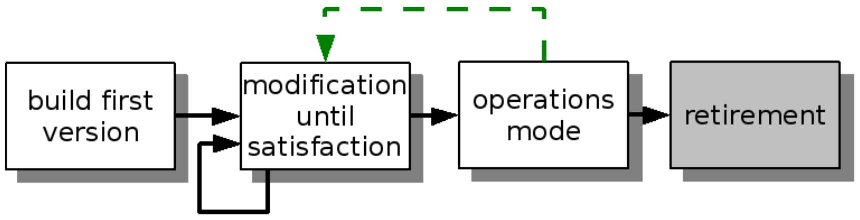


Figure 2.6: The build and fix model.

A specification only exists in the heads of customer and programmer which most likely do not match. b.) *no design* due to the lack of specification. The actual design happens through coding which is rarely documented and therefore only exists in the programmers head. Design errors will take immediate effect on the code and will be very expensive to undo later. c.) *high risk* since no specification and no design lead to a probably not satisfied customer. The project outcome is dependent on the goodwill of the customer, the programmer and the amount of money. Until the client is satisfied, the money runs out and/or the project is abandoned, the rework phase takes place which makes the project completely unpredictable in quality, time and cost. Finally, d.) *unpredictable costs* since the size of the project is not known. Costs cannot be predicted until the very late stage of the project, where they will most likely exceed the proposed sums.

2.5.2 Waterfall Model

The waterfall model is one of the simplest models where the phases of development are followed sequentially and transitions characteristically can only take place between two

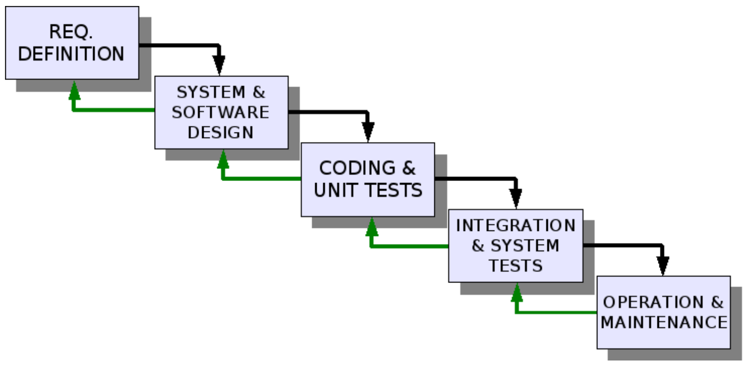


Figure 2.7: The waterfall model.

neighboring phases. There are various versions of that model depending on the application domain and the control flow between the steps (see Figure 2.7). Phases can be joined or new phases can be easily inserted. The process starts with the requirements analysis and definition. The design phase starts after completion of the first step and leads over to the implementation phase. When the implementation phase including unit testing is completed, the process carries on with the integration and system testing phase and if the system is successfully tested and installed the operation and maintenance phase is reached and active. If any defects are detected, there is only the possibility to step back to the previous phase for correction. Each phase must have a defined output, when a phase is completed the outcome is a product which mostly includes documents or reports like the requirements and design document or the test report and the user manual. The output of the implementation phase is the documented code with its unit tests. The waterfall model, although it was not called by its name then, was first introduced in the seventies [38] where several improvements of the original model are proposed. The waterfall model was actually developed as an improvement of the phase-model [39] which was an approach for the software development for the Semi-Automatic Ground Environment (SAGE) system of the United States North American Aerospace Defense (NORAD) in the nineteen-fifties.

The main advantage of the waterfall model is its simple nature. Tasks are split up into a sequence of clearly distinct phases with their own context. The waterfall model has a couple of limitations where the most severe are, a.) that the software requirements phase is complete when the design phase starts which is problematic for new systems because the end-user might not have a total understanding of the overall requirements of the system and therefore defects in the requirement propagate through until the system is finished which can lead to an almost unusable system, poor customer satisfaction and complicated maintenance. b.) For large software system the process lasts over years. With a completed hardware requirement the software will be designed for an old platform therefore requirements which might not be changed in a meaningful and controlled manner is unrealistic and not appropriate. c.) The software is delivered at the end of the process and the user has no idea what will be delivered. Furthermore, if the project runs out of money before

the implementation phase is finished, nothing can be delivered at all which imposes a high risk for the customer. Due to the document driven nature of the process the customer can end up with nothing but documents and no end product.
The waterfall model today has little practical value since it is too strict and rigid and only works well if projects are routine, the requirements are clear and fixed, the developers know the problem domain and the project is not too large.

2.5.3 (Rapid) Prototype Model

The prototype model eases the problems of the waterfall model such that before the actual requirements analysis is completed a prototype is made for the currently known requirements. This enables the customer to better understand the requirements of the intended end product. Valuable feedback can be gained from the prototype like a.) what is wrong, b.) what needs modification, c.) what is not needed d.) what is missing etc. The prototype itself undergoes a requirements analysis phase, a design phase, a coding and a careful testing phase (see Figure 2.8). Requirements which are well known must not be implemented to keep the costs and development time down since the prototype is thrown away and is not recycled. Although the costs of a prototype which is thrown away might be high, the costs of the following phases are decreased as well as the costs for late changes which are especially expensive. Advantages at a glance:

Requirements The requirements are collected and defined in a two stages process firstly through the normal analysis and secondly refined through the development of a throw-away prototype. This is especially useful for new systems where the end-user and customer has no clear idea of the requirements.

Predictable process Through the development of the prototype the overall complexity, time and hence costs of the end system can be more easily predicted.

Reduced Risk Through the refined requirements and the customer feedback from the prototype the risks of severe requirement changes is lower and therefore the overall risk for the project is decreased.

The disadvantage are the higher costs at the beginning and potential to make the mistake that well known requirements are implemented into the prototype as well as error and exception handling and that the code of the prototype is used for the product implementation. In the worst case the throw-away prototype model degrades to the build and fix model with all its consequences and negative effects on the costs and predictability. Moreover using this model it must be made clear to the customer that the prototype is indeed a prototype for finding and refining requirements and not an unfinished version of the end-system.

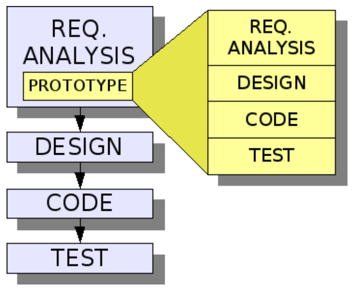


Figure 2.8: The throw-away prototype model.

2.5.4 Incremental Model

The incremental model is an evolution of the waterfall model and treats the developments steps as non discrete [37]. One can see it as a "multi-waterfall" cycle. Outgoing from the requirements the product is developed in functional increments (see Figure 2.9) where every outcome of an increment is an operational system. This technique begins with a simple initial (skeletal) implementation [40] and in every iteration, where the next functional increment is developed it is possible to easily react to changes which keep the costs down. This is possible since the requirements are broken up into smaller pieces which are less complex to understand and develop. A disadvantage of this approach is that the requirements must be well defined up front, as otherwise problems with the system architecture may arise during the iterations and this model degenerates to a build and fix approach.

2.5.5 Iterative (Evolutionary) Model

The evolutionary characteristics of software were already identified in the late 60s [41]. The iterative or evolutionary model can be considered as the most realistic of the tra-

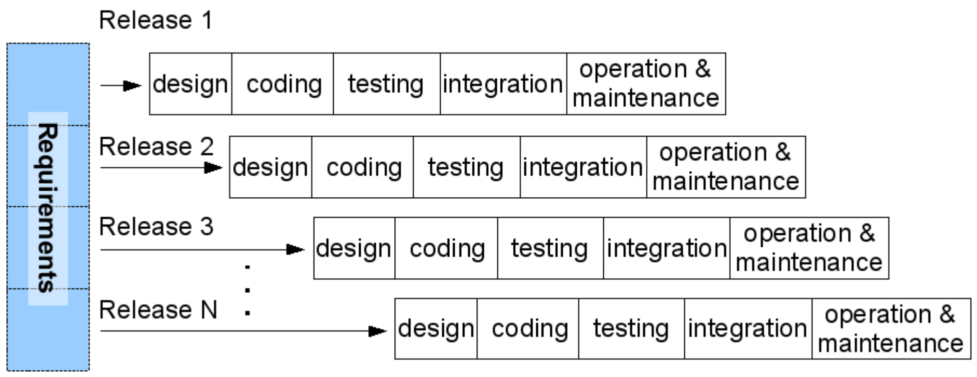


Figure 2.9: Incremental development model.

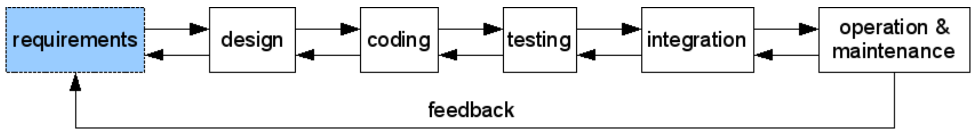


Figure 2.10: Iterative or evolutionary development model.

ditional software development models [37]. It is mostly used in combination with an incremental approach which leads to an iterative incremental development model. In Figure 2.10 one can see that each stage feeds back information to the previous and the collected gained insights after an iteration are fed back to the first step and even have impact on the requirements. In each iteration a usable and executable release of the software system is produced although not with all functional features and non-functional requirements until its final iteration. This approach is widely used, e.g., in the Unified Process 2.5.8 and in all agile methodologies in combination with the incremental method (see Section 2.5.9). Problems can arise through over iteration due to changing requirements and features which are identified throughout the iterations.

2.5.6 V-Model

The V-model [42] is a generic product model and adds validation counterparts to the early stages of the process (see Figure 2.11). In general the V-model is a waterfall model where the phases are arranged to emphasize the validation and verification and the relation between design and test steps. It is possible to adopt the phases to match any life-cycle. The model is split in two parts, where the left side represents the results of the development (production cycle) and the right side of the V shows the construction of the final product with the assembly of the individual components (testing cycle). Verification takes place at each step in the production cycle (left side of the V) to assure correctness of the step's outcome. Validation takes place at each step in the testing cycle (right side of the V) against the specifications of the left side and the actual implementation. The final test is the validation of the product against the user requirements. The V-model nicely shows the process of generating detailed specification starting with the user requirements down to individual modules or units on the left side which can be directly implemented and successively assembled to a system on the right side. It is possible to process parts of the V-model concurrently. Finishing a step on the left side the correlating step on the right side can be started, e.g., if the detailed design is finished the unit tests can be written so the test procedure is ready when needed. Left and right part of the V-model have almost always the same amount of steps (depending on the life-cycle) which shows that the test cycle also needs its time and must be considered in the overall process.

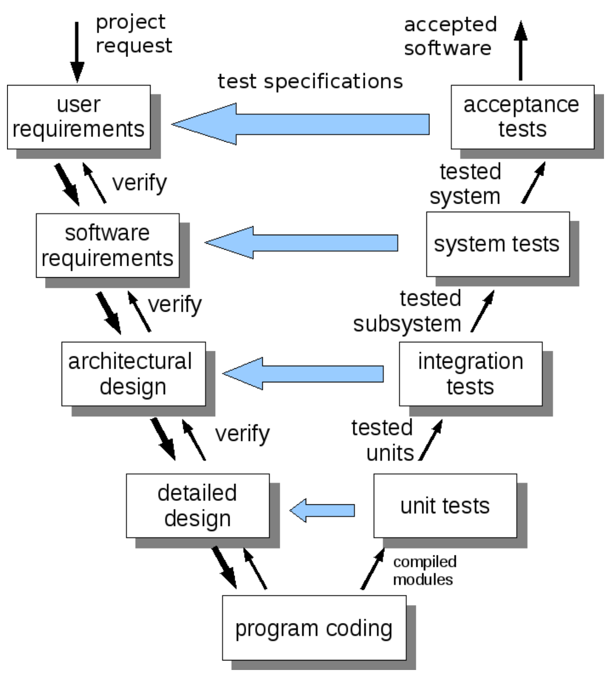


Figure 2.11: The V-Model.

2.5.7 Spiral Model

The spiral model [43] is an iterative model with four main phases that are organized like a spiral growing from the inside (close to the center) to the outside. The radial dimension represents the cumulative cost to date and the angular dimension represents the progress through the spiral and hence the progress. A cycle starts with the determination of the objectives, its alternatives and constraints. Finishing this determination step the alternatives, risks and solutions are evaluated. Especially the risk and its possible solutions are emphasized and with techniques of prototyping, simulation and benchmarking, strategies are worked out to overcome these risks. If no solutions for risks can be found, the project can be terminated at this stage. These activities lead to the next phase, the development and verification of the next level product. Depending on the proximity to the center and therefore the number of iterations, either the concept of the operation, or the actual requirements, or the software design is created. In later iterations the implementation testing, validation and verification also takes place in this phase. The last phase in the cycle is the planning phase for the next iteration where near the center in an early iteration the requirements and the life-cycle plan is created as well as the development plan, the integration and test plan in later iterations (see Figure 2.12). Through the use of prototypes the overall risk is minimized. The spiral model in its simple form is a kind of consecutive waterfall model preceding each phase with an alternative risk analysis and follows each phase with an evaluation and planning of the next step. The weakness of this model is that it was designed for big projects and large-scale software since the risk

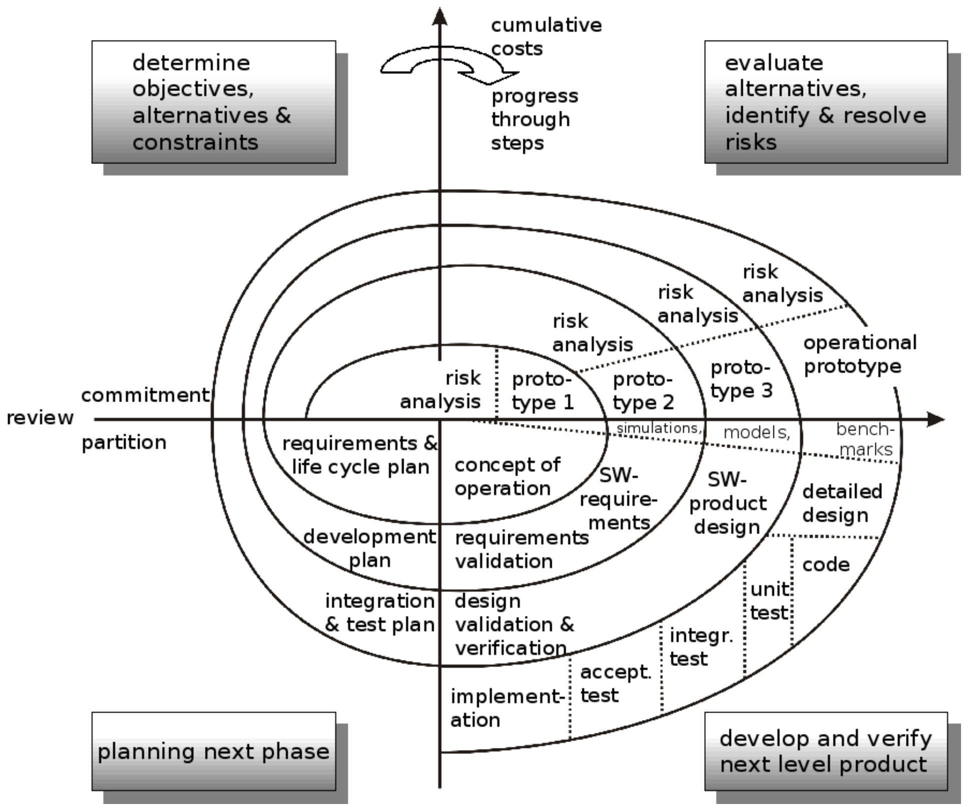


Figure 2.12: The spiral model.

analysis would be too costly for small software projects. It is better suitable for in-house development, otherwise all risk analysis must be made before the project is launched and the contract is signed. Advantages at a glance:

Risk analysis Through the risk analysis which takes place in each iteration, it is no problem to determine what and how much should be tested.

Maintenance Maintenance is integrated in this process model. An additional iteration is a full maintenance cycle.

2.5.8 Unified Process (UP)

The Unified Process is an iterative and incremental software development framework (all activities can be customized according the project needs [44]) for dealing with the whole life-cycle of a software project. In the Unified Process the communication with the customer and the method to describe the system from the customer's view by means of use-case diagrams is highly important. The importance of software architecture is, that it emphasizes and "helps the architect focus on the right goals, such as understandability, reliance to future changes, and reuse" [45]. In the general implementation the UP defines who does what, when, how and what goals are to be reached as well as which inputs and outputs for these activities are expected. This seems pretty heavy-weight but since it is a framework it can be customized to fit the need of agility and even XP. There are several derivates of the Unified Process, e.g., OpenUp [46], Agile Unified Process (AUP) [12], Enterprise Unified Process (EUP) [47] and the probably most known, the Rational Unified Process (RUP) [48]. There are four key characteristics backing the UP [49]: it is a.) iterative and incremental, b.) use-case driven, c.) architecture-centric, and d.) risk acknowledging. The UP has the following different project life cycle phases: a.) inception phase $\Rightarrow$ vision, b.) elaboration phase $\Rightarrow$ baseline architecture [50], c.) construction phase $\Rightarrow$ complete beta release and finally the d.) transition phase $\Rightarrow$ final release. The five different disciplines in the development process (requirements, analysis, design, implementation and testing) are not worked through sequentially in the UP but can be involved in different phases or stretched over more than one phase (see Figure 2.13 [49, 13]). The disciplines are steps which are followed but in the various phases there are different major targets, e.g., in the inception phase mainly the requirements finding and definition takes place, although there are already some analysis activities. The main focus is the outcome of the requirements definition and more detailed analysis will take place in the elaboration phase. The design, implementation and testing are even broken up into chunks in a way that they are partly active in different phases where necessary but again clearly have their main focus in their discipline specific phase (see earlier description of phases). A discipline is a set of activities which must be done to achieve the goal of the discipline. It tells what is to be done, what is the input, and what the expected output. The Unified Process applied to a project means that this framework must be adapted to the project. There is no such thing like a universal process which fits every project. A process is always highly project and organizational dependent and relies on experiences. The UP is an elaborate framework which is flexible and can be trimmed to the projects needs. It even can be modified to meet agile needs and where necessary elements can be omitted or enhanced.

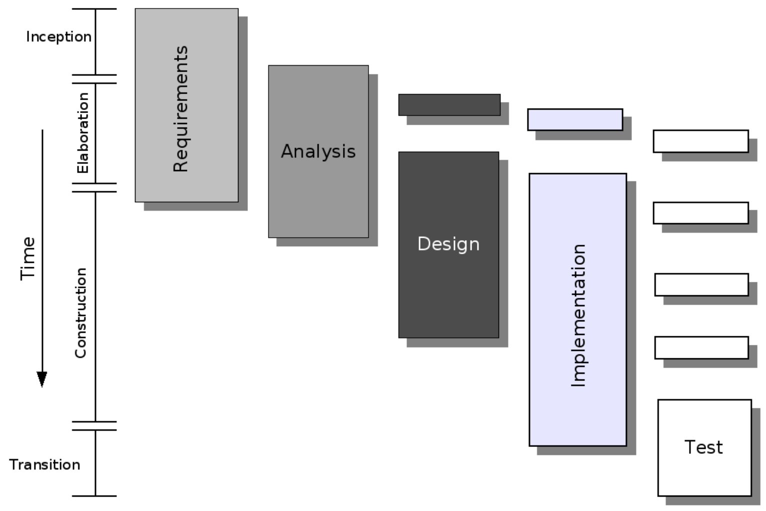


Figure 2.13: Unified Process phases vs. disciplines.

2.5.9 Agile Methods

Agile methods take up a weakness that all the above, more traditional, SW development methods share: namely their slow reactivity to requirement changes. Although the starting point for the "agile" development movement can be seen as the time of introduction of Extreme Programming (XP) [51], the roots for the agile development methods can be found way back in the times where the fiction of universal methods still existed - in the late nineteen-eighties. In Figure 2.14 an overview of the agile evolution [7] can be seen. The boxed methods which are identified as agile are depicted as boxed, whereas those methods and authors which had a direct influence to the manifest are marked with dashed arrows. According to Jim Highsmith [6] agility is the ability to balance flexibility and stability. The word "agile" means "quickness", "lightness", and "ease of movement"; "nimble" or "mentally quick" or "alert"[2]. This implies quick response to emerging changes which is in fact an important issue in today's software development practices in general and especially in the volatile Internet and mobile device application development. With process oriented and plan driven software development models [8] one of the key features is the determination and fixation of the requirements before the design and actual development phase starts. However it is questionable if in practice the rules of the various development models are followed strictly and thoroughly [52]. Furthermore, it has been shown that certain plan driven methods are not necessarily suitable for all people [53] or settings [54]. For this reason, an individual personal development process has been proposed [55, 56]. Although problems obviously exist, software is still widely developed and shipped using

[2]http://www.thefreedictionary.com/agile - 2007

traditional development models or no models at all [57]. A group of seventeen consultants and industry software developers formulated the Agile Manifesto [1] which was signed in Utah in February 2001.
The Agile Manifesto expresses the core philosophy of agile software development.

Individuals and interaction over processes and tools ⇒ people matter. The first focal topic is about the human role of each individual and the interactions between people in the development team.

Working software over comprehensive documentation ⇒ working software is the main point of all agile practices which is to be delivered in short cycles and frequently - rather months than years and rather days than weeks.

Customer collaboration over contract negotiation ⇒ the client is not the enemy. Nevertheless contracts are important. The cooperation and collaboration between the developers and customers is vital since the product should satisfy the customer and not only fulfill the contract.

Responding to change over following a plan ⇒ the customer as well as the development team should be authorized to decide about adoptions which emerge in the development process. Therefore this is all about communication and information.

Agile methods are not about how to avoid changes in a project but instead how to cope with the unavoidable changes throughout the whole product life-cycle [3]. Therefore agile methods are designed to a.) deliver the first release as soon as possible to get early feedback, b.) keep everything small and simple so changes can be done easily, c.) constantly better the design to ease the next iterations implementation and decrease the costs and finally d.) constantly test for early defect recognition, neutralization and documentation. All in all it is about minimizing the project risks. Table 2.3 compares the process-oriented with the agile methods [8]. The underpinning and more detailed principles of agile methods, along with explanation adding more information to the coarse direction of the manifesto points can be found at [1].
There have not been many studies about return of investment into different process methodologies and less about using agile development methods. There is not really a short cut or easy method to determine which model guarantees the largest benefits for the current project in the given environment. Therefore it is still a matter of personal judgment and experience to choose the right process model. There are a number of different agile methodologies, briefly described below, which have been (re)invented and made widely be known since the introduction of XP. First off, it should be mentioned that the Adaptive Software Development and Feature Driven Development methods cover both project management as well as development process aspects, whereas Extreme Programming does not cover detailed project management aspects and Scrum offers no instructions on the design-, coding and testing phase in detail.

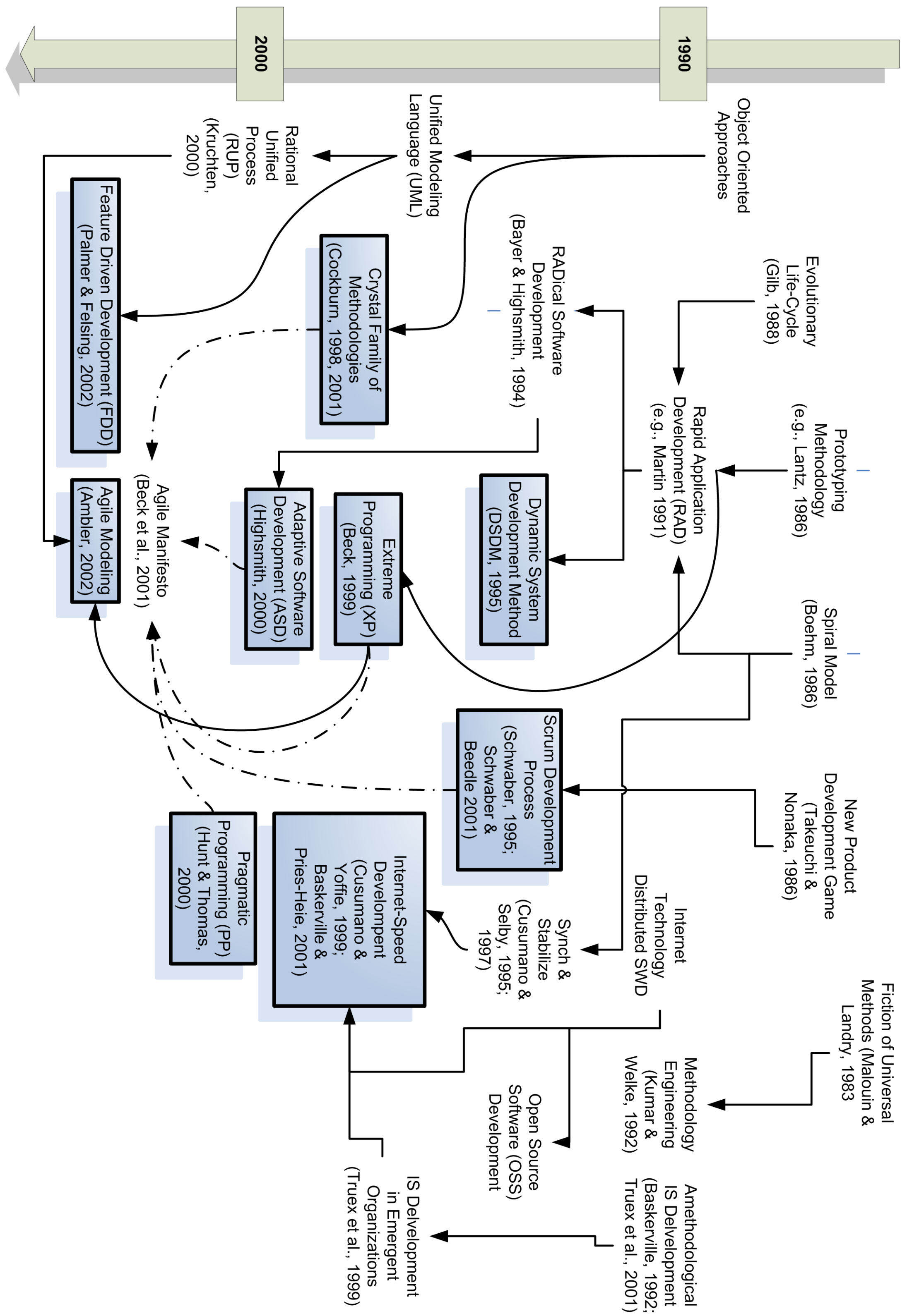

1990
2000
Object Oriented Approaches
Unified Modeling Language (UML)
Rational Unified Process (RUP) (Kruchten, 2000)
Evolutionary Life-Cycle (Gilb, 1988)
RADical Software Development (Bayer & Highsmith, 1994)
Crystal Family of Methodologies (Cockburn, 1998, 2001)
Feature Driven Development (FDD) (Palmer & Felsing, 2002)
Prototyping Methodology (e.g., Lantz, 1986)
Rapid Application Development (RAD) (e.g., Martin 1991)
Dynamic System Development Method (DSDM, 1995)
Extreme Programming (XP) (Beck, 1999)
Adaptive Software Development (ASD) (Highsmith, 2000)
Agile Manifesto (Beck et al., 2001)
Agile Modeling (Ambler, 2002)
Spiral Model (Boehm, 1986)
New Product Development Game (Takeuchi & Nonaka, 1986)
Scrum Development Process (Schwaber, 1995; Schwaber & Beedle 2001)
Internet Technology Distributed SWD
Synch & Stabilize (Cusumano & Selby, 1995; 1997)
Internet-Speed Development (Cusumano & Yoffie, 1999; Baskerville & Pries-Heie, 2001)
Pragmatic Programming (PP) (Hunt & Thomas, 2000)
Fiction of Universal Methods (Malouin & Landry, 1983
Methodology Engineering (Kumar & Welke, 1992)
Open Source Software (OSS) Development
Amethodological IS Delvelopment (Baskerville, 1992; Truex et al., 2001)
IS Delvelopment in Emergent Organizations (Truex et al., 1999)

Figure 2.14: The evolution of agile methods.

Home-ground area	**Agile methods**	**Plan-driven methods**
Developers	Agile, knowledgeable, collocated, and collaborative	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Knowledgeable, collaborative, representative, and empowered
Requirements	Largely emergent; rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primary objective	Rapid value	High assurance

Table 2.3: Home ground for agile and plan-driven methods [8].

Extreme Programming (XP)

Extreme Programming is one of the most widely known and popular form of agile methodology. The main focus of this method is development and not the management of the project. Most of the practices which were made prominent through XP are acknowledged in the agile community and widely adopted by other agile development methodologies. The underpinning or “driving” values [20] of XP are:

Communication In every development process (not necessarily a software development process) communication is crucial. Whatever reasons for poor communication exist, many problems or defects within software systems can be traced back to poor communication during the development of the system [13].

Simplicity In XP it is tried to always find the simplest solution that might possibly work and is appropriate for the current problem, therefore it is easier to understand, implement, test, debug, correct and refactor.

Feedback Good feedback is essential and it is even better when it comes in time. Feedback is needed early and as often as possible from the different involved parties like the customer, the end-user, etc. so refinement of the requirements can take place early in the process as well as defects and other problems can be dealt with at an early stage or at least when they are recognized. This value strongly correlates with the first one.

Courage To adhere to all these values throughout a project, courage is needed. For all the activities like refactoring, dumping useless code, keeping things simple, communicating in an honest straight fashion, and convincing the management and the customer to run a project in XP mode with all its features, courage is needed.

Respect Team members must respect each other's work. If there is no mutual respect or members do not care about the project the project will not work out at all. For software development to simultaneously improve in humanity and productivity, the contributions of each person on the team need to be respected [20].

The twelve core practices, as described in [51, 58], are characteristic for the Extreme Programming methodology and effectively define the process (see Figure 2.15).

1. **Planning Game** - planning of the next release is done within a so called "Planning Game" which is an incremental process. At the project start a crude overall plan is made which includes the determination of the overall functionality as well as what functionality will be realized in the different releases and when they will occur. At the start of every iteration, details about this iteration or release are determined and the implementation is planned for the release.

2. **Small Releases** - it is important to determine the minimum release cycle which adds business value to the system. The advantage of small releases is that the end-user is frequently provided with a working system and hence can give accurate feedback to the development team which makes short term planning easier so in the planning game only the next weeks are considered and deviance from the plan is limited.

3. **Metaphor** - the system metaphor is a story or view that expresses the overall way in which the system will operate [13]. It is intended to provide an overall abstraction of the system's functionality, its purpose, what it does, how the parts fit together similar to the architecture of a system (e.g., in the Unified Process 2.5.8) but in a much briefer and less detailed manner.

4. **Simple Design** means that it meets its needs and does nothing more. The programmers must design the system for the current iteration. Features which are not used in this iteration are potentially not used in later iterations as well. Therefore it is very important to keep the design as simple as possible satisfying the current needs. XP offers a guideline how to characterize a design as "simple" - it must a.) pass all available tests, b.) must not have duplicate logic, c.) states every intentions important to coders and d.) has the minimum possible classes and methods [58].

5. **Test First** means that test are written before the actual code is implemented. As soon as the requirements are defined acceptance tests are created therefore test code for every implemented code is produced. Through the test-first approach it is easy to produce implementation code that is as simple to satisfy these tests as well as what the code should do. Furthermore refactoring is made easier since side-effects are detected as soon as the previous tests do not succeed any more.

6. **Refactoring** - with the term “refactoring” the change of a system’s implementation is meant without changing its functionality. This is mostly used to simplify coding constructs to make it more understandable, e.g., after a new feature has been added to the implementation or to change the code for the sake of enhancements, so that new features can be implemented more easily.

7. **Pair-Programming** was proven to have the benefits of better code quality and lesser defects [59]. Furthermore knowledge transfer between the team members during their pair-programming session takes place and hence extra tutoring for, e.g., new or junior team members can be saved.

8. **Collective Code Ownership** means that whoever notices code which can be simplified, not only is allowed, but is behold to simplify it. Of course for collective ownership everybody in the team must know what the code does in all areas so as a prerequisite the knowledge transfer, e.g., done trough pair-programming, is essential.

9. **Continuous Integration** means that if a task is finished and all its specific tests run through it is integrated into the current build. The integrated code must still pass all previous as well as all newly introduced tests to be accepted for check-in and a creation of a new build. When problems occur during integration code must not be released to the versioning system until the problems are solved. This procedure should take place at least once a day but of course can be repeated more often and can easily be supported by automatic builds of the system.

10. **40-hour Week** is more a guideline to have an appropriate amount of working hours per week. Of course it can happen that there is a need for over-time but it should not be the general case. People work better and make less mistakes if they are relaxed and not under constant pressure working every weekend up to 80 hours and more - even isolated overtime if happening frequently is a sign of severe project problems [58]. The exact number of hours are not important but to keep the amount the team works on the project to a limit keeps everyone “fresh and eager” [58].

11. **On-Site Customer** is meant to have the information of the customer at one’s fingertips. It is most likely not possible to have a real customer on the team all time but to minimize the risk of coding without knowing exactly what the customer needs, it is vital to have a tight relationship and constant feedback meetings with the customer - the one who really will use the product - to be sure to be on the right track.

12. **Coding Standards** are needed when more than one person has to read, understand and enhance the code - which is a fact on an XP team. Especially through collective code ownership, pair-programming and refactoring it is a must to have a simple coding standard which needs the least amount of work possible [58] to adhere to it and ensures that the code is easily understood and accessible by any programmer of the team.

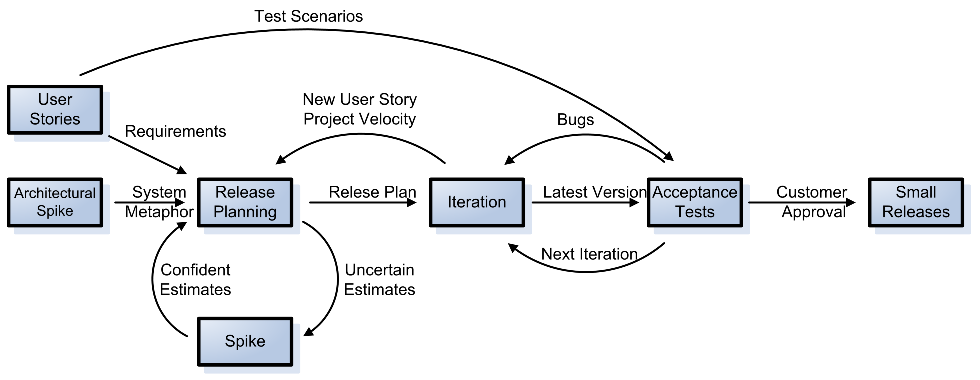


Figure 2.15: The XP process.

Even non-agile development can benefit from adopting single AM practices, since they are based on many discussions and have been tested in real-life projects. Adopting one or more practices from XP does not necessarily imply that the development process becomes an XP process or an agile one at all. To do so all practices must be implemented, tried out in practice and later on can be adopted if needed, otherwise one cannot call a process XP. Extreme programming is evolving over time therefore the practices slightly changed with growing experience from practical applications and probably will change in the future further if necessary.

Adaptive Software Development (ASD)

Adaptive Software Development [60] was introduced in the year 2000 and is based on the theory about adaptive systems and previous research on iterative development methods, e.g. “Radical Software Development” [61]. The ASD mainly focuses the problems in development of large complex systems. Therefore it fosters incremental and iterative development with constant prototyping. ASD consists of a three-phase cycle a.) speculation (planning) phase, b.) collaboration (development) phase and c.) learning (review) phase. The ASD life-cycle can be seen in Figure 2.16 where the actual learning loop is bordered by the project initiation and the final quality assurance and release phase which are not part of the cycle and are only stepped trough once. The ASD is mission-driven which means that the whole project and especially the development is carried out to complete the so called project mission which is defined in the initiation phase and represents a coarse description of the end product. Aim of the mission definition phase is to find the information that is necessary in order to finish the project successfully. This is laid down in three documents a.) the project vision charter, b.) the project data sheet and c.) the product specification outline [60]. Furthermore the overall schedule and the development cycle are defined in the initiation phase. The development cycle is component oriented which means that results and quality count more than the process which leads to the result. Typically, it lasts between four and eight weeks. Out of repeated quality reviews,

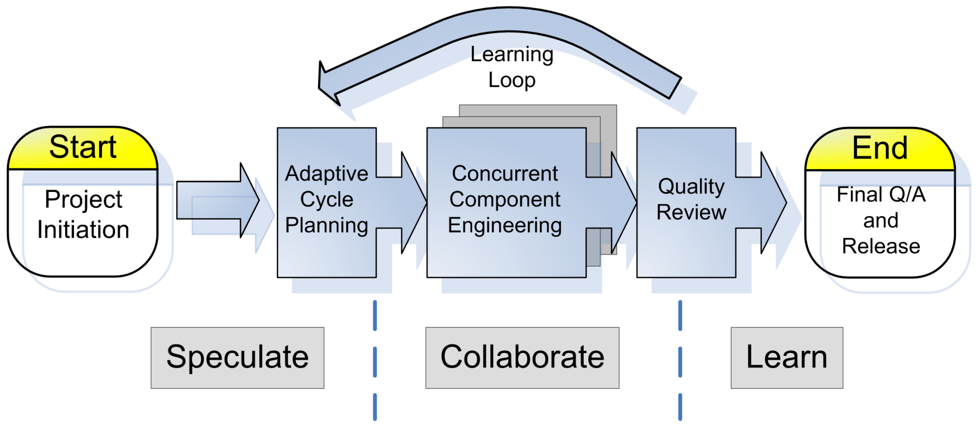


Figure 2.16: The adaptive software development process.

information is retrieved for the learning cycle to improve the functionality and quality of the product. This method also describes a model for project management which is called Adaptive Leadership Collaboration Management model [60].

Dynamic System Development Method (DSDM)

The Dynamic System Development Method[3] is not specifically designed for software development, but is a dynamic and modular framework which focuses on development of a whole system. Nine principles [10, 11] are essential to handle a project in DSDM manner otherwise project risks are increased significantly. The principles of DSDM are a.) active user involvement is mandatory, b.) the team must be empowered to make decisions, c.) the focus is on frequent delivery of products, d.) fitness for business purpose is the essential criterion for acceptance of deliverables, e.) iterative and incremental development is mandatory, f.) all changes during development are reversible, g.) requirements are base-lined at a high level, h.) testing is integrated throughout the life-cycle, and finally i.) collaboration and cooperation between all stakeholders is essential.

In Figure 2.17 the overview of the DSDM process model is depicted with its seven distinct phases (the sub-stages of the three iteration cycles are omitted but briefly described later in the text).

Pre-Project phase Project suggestions, funding and commitment are evaluated and a final decision about the project realization is accomplished.

Feasibility Study Technical feasibility, their potential costs and risks building the system using DSDM are considered and examined as well as the tasks of the system are defined which satisfy the requirements for the business needs.

[3]http://www.dsdm.org

Business Study The insights of the feasibility study are enhanced and the user requirements and requirements in general are examined and prioritized (in accordance with the stakeholders) as well as the processes are determined which are affected and influenced.

Functional Model Iteration A functional model is created leading to a development of a prototype. The prototype is constantly reviewed by different user groups during this iteration. This phase consist of four sub-phases a.) identifying the functional prototype - work out the functionalities to implement, b.) agree on a schedule - who, when and how these functionalities are to be developed, c.) create the functional prototype - development of the prototype, d.) review of the prototype using the test results of the test team and the testing user. The results are compiled into the review document.

Design and Build Iteration The system is engineered iteratively. Testing is vital in this iteration to get feedback for further iterations. This iteration has four sub-phases a.) identifying the Design Prototype - functional and non-functional requirements are determined for the integration step to a testable system, b.) agree on a schedule - who, when and how these requirements are realized, c.) create the design prototype - the system is integrated ready for employment in daily use, d.) review the design prototype - the system is verified and validated. The outcome of these tests is compiled to an end-user documentation and serves as knowledge for refinements for further iterations.

Implementation The working, tested and approved system (including documentation) is put into operation. The Implementation phase has four sub-phases a.) user approval and guidelines - the system is approved according the requirements; user guidelines are created, b.) train the users on the system, c.) implementation - the system is implemented in the operational environment, d.) review business - the practical daily use is evaluated whether it meets the requirements of the business case. It is decided whether a further iteration in any of the previous phases is needed or the project can be finished technically.

Post-Project Phase It takes place after the project has technically finished and the system is used in practice over a considerable period, e.g., six months. An evaluation concerning the performance, efficiency and effectivity of the system as takes place as well as collecting requirements for potential enhancements and maintenance are accomplished.

Not all of the phases are mandatory but are subject to potential omission due to adoption to special project needs. Only the nine principles must be followed and implemented - if this is not possible for one project then the DSDM approach is probably not adequate for this project.

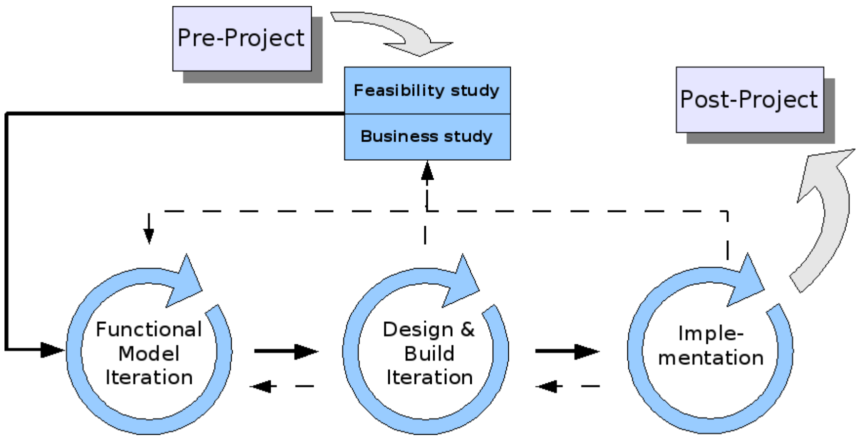


Figure 2.17: DSDM process model.

Scrum

Scrum is not limited to software development but is meant to be a general framework for leading product development. The term “scrum” comes from a game strategy in rugby where it means “getting the ball back into the game” in teamwork [4]. Scrum was developed to manage the complex and unpredictable process of system development and does not define techniques for software development but rather focuses on the team members work to react properly and flexibly to the permanent changing environment. Scrum is intended to improve existing practices and discover any problems or deficiencies in the development process and their used practices. Scrum is similar to FDD since at the beginning of the project the requirements are determined and collected in the product backlog (see process overview in Figure 2.18) and efforts are estimated. From this pool of functionalities, which all have to be implemented, a subset is chosen for the upcoming sprint (iteration) and is moved to the sprint backlog. These features are chosen according to the priority for the customer. The sprint backlog is processed within a fixed time, e.g., a month where no changes occur in the sprint backlog but if necessary in the product backlog. This ensures that the developers can do their job without interruptions during the sprint, since changes can only be demanded between sprints. At the end of the sprint a sprint review meeting takes place and the next iteration is planned. In this review meeting new backlog items might emerge as well as a complete change of the direction the system is to be developed. Project management is done by the Scrum Master who is in constant information exchange with the customer and the development team. Therefore, every day a meeting called “Daily Scrum” [10] is held where the developers talk about the current sprint, the achievements and the existing problems to keep track of the projects progress. The Scrum Master’s role is more that of a coach than the manager of the project since the team is self organized and decides what to do. Scrum is a method for small teams up to 10 people - if more developers are involved in the project multiple scrum teams between 5-10 members should be formed [4].

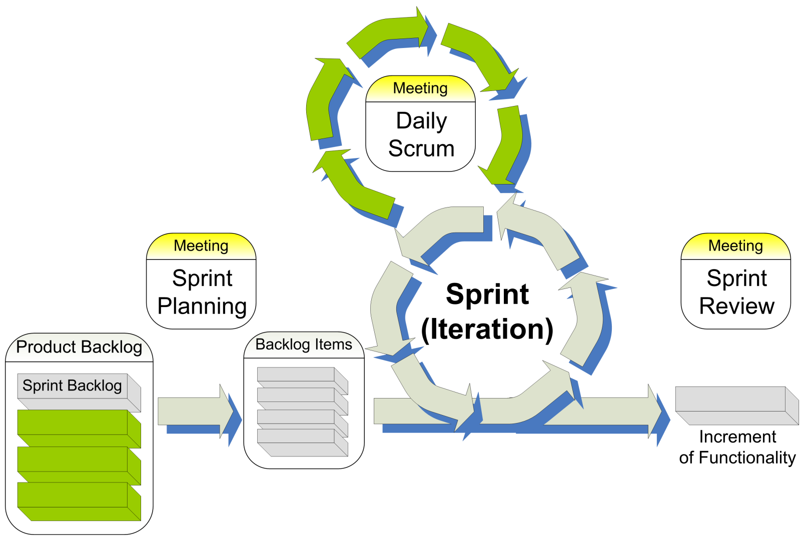


Figure 2.18: The Scrum process overview.

Crystal

Crystal is a set of methodologies taking into account that different projects need different methodologies in terms of, e.g., the security level of a web game versus a nuclear power plant control system or software in public transport. Therefore the aim is to select the most suitable methodology of the crystal family for each individual project. The Crystal approach is to collect concrete sample methodologies that already have been used on projects and tune them on the fly to fit the different project-environments and circumstances [2, 62]. In Figure 2.19 a part of the Crystal family matrix is depicted. The horizontal dimension represents the size of the project in terms of the number of people who must be coordinated corresponding with a darker color and indicating a heavier methodology. The vertical dimension represents the system's criticality, the strictness of the methodology and the potential impact of a system failure. The levels of criticality are "C" for comfort, "D" for discretionary money, "E" for essential money and "L" for life [2]. The two restrictions on the Crystal methodologies are, a.) that they do not currently exist for life-critical systems (see Figure 2.19) and b.) they are based on teams sharing the same location. Furthermore, Crystal is not intended for a criticality level of "essential money" (see the gray shaded square in the clear column of Figure 2.19) but it can possibly be stretched by the team to fit to this level which increases the effort of strictness and discipline of the process. Crystal tries to define the least disciplined process that could possibly succeed which increases the potential that every team member will follow the methodology since the stricter and more disciplined the methodology the less likely it becomes that everyone will follow the rules. That means that in this methodology the success of the project weighs more than efficiency (costs and time savings through a more disciplined process) [2].

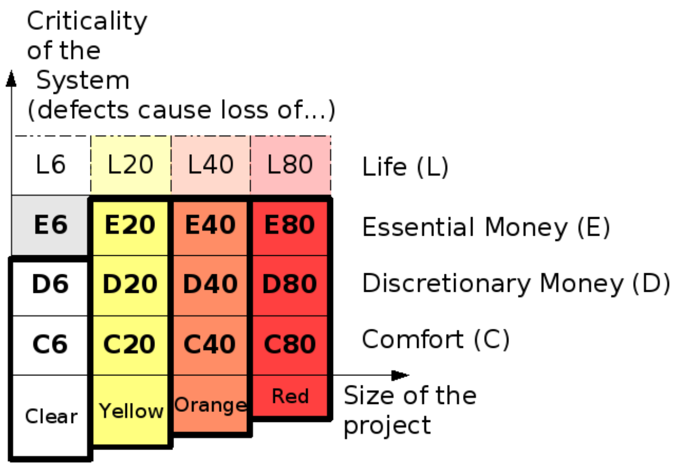


Figure 2.19: The family of Crystal methodologies (named by color) [2].

Feature Driven Development (FDD)

Feature Driven Development was first written about and given a name in 1998 [14]. The FDD approach does not cover the entire software development process but emphasizes the design and building phase [63]. It differs insofar from all other agile methods that upfront a detailed model of the project is created. In Figure 2.20 the general five process steps of the FDD can be seen. In the first phase a high level description of the system is created and the overall domain is split up in smaller domain areas. After this phase a comprehensive list of features is retrieved which are planned to be implemented one at a time. The feature lists are reviewed by the customer and end user. A feature is small in size and it should be possible to implement a feature in a few hours or a day. In the “plan by feature phase” it is decided which feature is implemented at what time (prioritization) whereas each feature itself is iteratively planned in detail. The planning in each iteration is necessary since changing features can have side effects and impacts on other features which have to be taken into account and refactored. The “design by feature” and “build by feature” phases are depicted in detail in Figure 2.21. During the development of the features constant inspections and reviews take place as well as continuous build cycles. A small group of features is selected from the feature list. An iteration can typically last between two days and two weeks. The features can be designed and built concurrently by different teams according to the domains. The iteration includes design, design inspection, coding, unit tests, integration and code inspection. At the end of an iteration the successful implemented features are moved to the main build whereas the iteration continues with another group of features from the list. The actual project progress is measured according to implemented features and their estimated relevance to the project to prevent the 90% syndrome (90% of the project being 90% complete 90% of the time) [10]. In contrast to XP for each feature a feature owner [14] exists who is responsible for the entity according to the statement “Everyone can’t know everything about everything” [64].

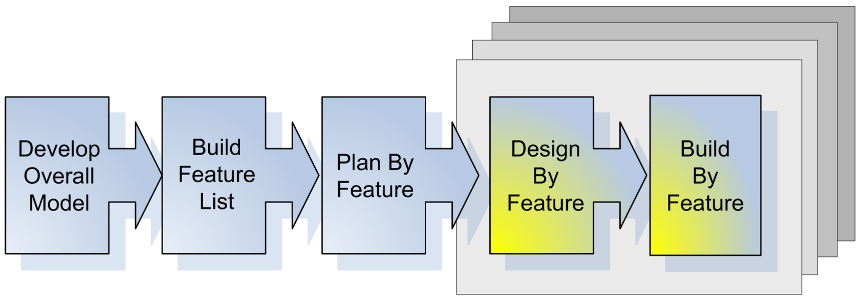


Figure 2.20: The Feature Driven Development process overview.

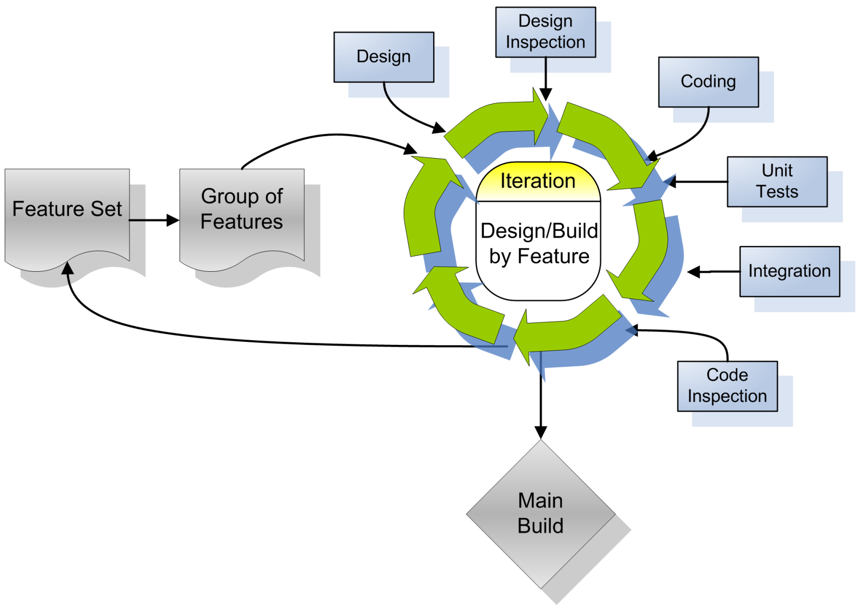


Figure 2.21: The FDD Design & build by feature process.

Agile Modeling (AM)

Agile Modeling is not a complete methodology but rather an approach modeling aspects of software development methods like the Unified Process or Extreme Programming. It is an approach that "aims to model just enough and no more!" [13]. Agile models are intended to

Provide positive value AM is not the end in itself but the support to the development process therefore there is no need to model every detail of the software system - modeling every aspect is like programming the software in another language which is a waste of resources. The outcome of AM is an abstract model of the software system so its purpose is to communicate the general functionality of the software, how the building parts fit together and work. If these requirements are fulfilled the model is detailed enough and provides additional value.

Fulfill their purpose Modeling must be done on purpose not because the process prescribes a model to be created. The purpose includes the level of detail. Different models can be created to explain the functionality of the same software system in different level of detail to different people, e.g., to customers, executives or project co-workers. If the purpose is not clear then a model might not be needed.

Be understandable The models must be understandable for those who they were created for. Simplifications and abstractions can make the model more compact but can hide necessary details - it depends on the audience.

Be sufficiently accurate Sufficiently accurate means that there is no need to cover every aspect in detail but it still must be sufficiently understandable by the intended audience, sufficiently detailed but as simple as possible.

Be sufficiently consistent Sufficiently consistent means that although a model might become slightly inconsistent due to refactoring it must keep the ability to transport its message to the intended audience. If this is not the case it must be adapted to become understandable again.

Be sufficiently detailed AM is supposed to be lightweight which means that only a level of detail is incorporated in the model the audience needs to see. Here the agile axiom "YAGNI" (You Ain't Gonna Need It) is applied since many models are built in too much detail. An agile model is created with the minimum level of detail which is required by the audience.

Be as simple as possible Agile models apply to the inherent agile fundamentals "KISS" (Keep It Small and Simple) and "YAGNI" and it is tried to use only a minimum of detail and complexity of syntax in the model.

In Agile Modeling there is another maxim which says that "content is more important than presentation". Therefore it does not matter if the model is drawn by hand as long it fulfills its purpose. The purpose also defines which kind of model is used, e.g., class, sequence, collaboration, activity diagram or simply a flowchart - whatever fits best to convey the information to the intended audience. There are also no rules whether to use CASE (Computer Aided software Engineering) tools or not and whether the use of UML (Unified Modeling Language) is sufficient which can be difficult in, e.g., GUI or data modeling. Finally, AM tries to apply the agile principles and tries to support the development process and is not meant to be a method on its own.

Chapter 3

Analysis of the htmlButler Project

In this chapter the htmlButler project, a small sized academic project which was conducted from February 2005 until February 2007, is analyzed both from a technical and a project management viewpoint. First, an overview of the project goals are given, then architecture and a use case is described and finally the post-project feedback of the involved developers is incorporated into the project analysis. Section 3.1 until Section 3.2 is a consolidated complete revision of [65, 66].

3.1 Design and Implementation

The htmlButler project aimed at enhancing the usability of visual wrapper technology for retrieving information on a regular basis. It should allow an untrained user to visually specify simple wrappers. More tech-savvy user should have been able to specify more complex wrappers through enhancing the underlying ontology. Users could create wrappers by interacting with the htmlButler server through a standard web browser or by using a browser plug-in. It was planned that a) the user could choose between having the wrapper being executed at regular intervals on the htmlButler server, or on their own computer in their browser plug-in, b) wrappers could be shared between users because of the central storage of previous generated wrappers and c) users could alter wrapper definitions or contribute further semantic concepts while wrapper creation, thereby growing the overall ontology and acting as a natural feedback system keeping up an acceptable quality level. htmlButler is based on some ideas of the Lixto set of tools [67, 68] that allow application developers to implement wrappers[1] without the need for manual coding. Nevertheless most tools like [67, 70] have a steep learning curve. Others may lack an easy user interface. Although, with those tools many options are available and sophisticated wrappers and aggregations can be created, they are not so well suited for the "normal" Web-user who has no knowledge about the wrapping technology in detail, protocols and data structures on the Web.

[1]Wrappers are specialized program routines that automatically extract data from Internet web sites and convert the information into a structured format [69].

3.1.1 Usage Scenario

The user enters the URL of the tourism portal and clicks the submit button (see Figure 3.1). An HTML frameset is loaded into the client browser where the upper part shows the htmlButler options and the lower frame holds the actual tourism portals homepage. In the lower frame all links can be clicked and it is possible to navigate as usual. This is made possible through the transparent htmlButler proxy that redirects all links, cgi-calls and form submissions so that the wrapper can be evaluated in the proxy and stored for replay after user approval. Every page the user navigates is analyzed by the htmlButler system - the text content of this HTML page is stripped from stop-words, the remaining text is stemmed by a porter-stemmer and according to the actual calculated TFIDF (term frequency inverse document frequency) value the page is categorized into one of the system stored categories. The user can accept this suggestion or change it in the drop-down list and update the status by clicking the update-button. The system updates this status and remembers the user decision. The user navigates to the desired *“Super Last Minute”* page by clicking on the appropriate link at the top of the portal page in the lower frame (see Figure 3.2). If the user is interested in being notified about changes for a particular flight, he selects the row or a part of it in the table and clicks the *Next*-button in the upper htmlButler frame (see Figure 3.3). The selection is submitted to the htmlButler server, tokenized according the concepts stored in the ontology of the category which was suggested by the system or chosen by the user. An HTML page containing the possible options for each recognized token which triggers notification is delivered to the client browser’s upper frame (see Figure 3.4). In Figure 3.5 the changes the user made are shown, e.g., the option of the status token was set to *ignore*, the destination- and the start-airport are set to *”exact match”*, the date, time and duration are set to *”any change”* and finally the price is set to *”up to this amount”* and the value itself is changed from 198€ to 190€. These settings mean that a notification is sent to the user if anything changes for the time, date or the duration token or if the price drops to 190€ or below, for the chosen destination and origin. After clicking the *Next*-button the options are sent to the htmlButler server and the wrapper is configured with these options. Finally, the user must provide his email address (see Figure 3.6) It may also be that the user is recognized by the system from previous wrapper generations via cookies and the field was pre-filled with the known email address. The user can of course change it if it is not appropriate. After clicking the *Next*-button, the system responds with an email confirmation as a reference to the user. Had the user not been recognized via a cookie, this mail would have additionally contained a link the user would have to click to authenticate the given email address in order to activate the wrapper. The user can then proceed to define another wrapper.

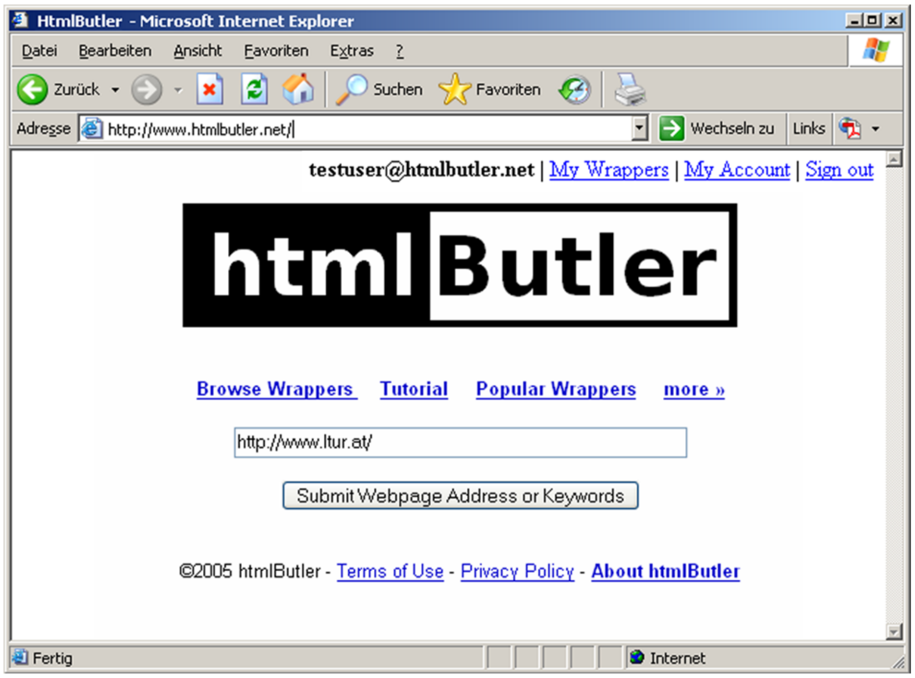
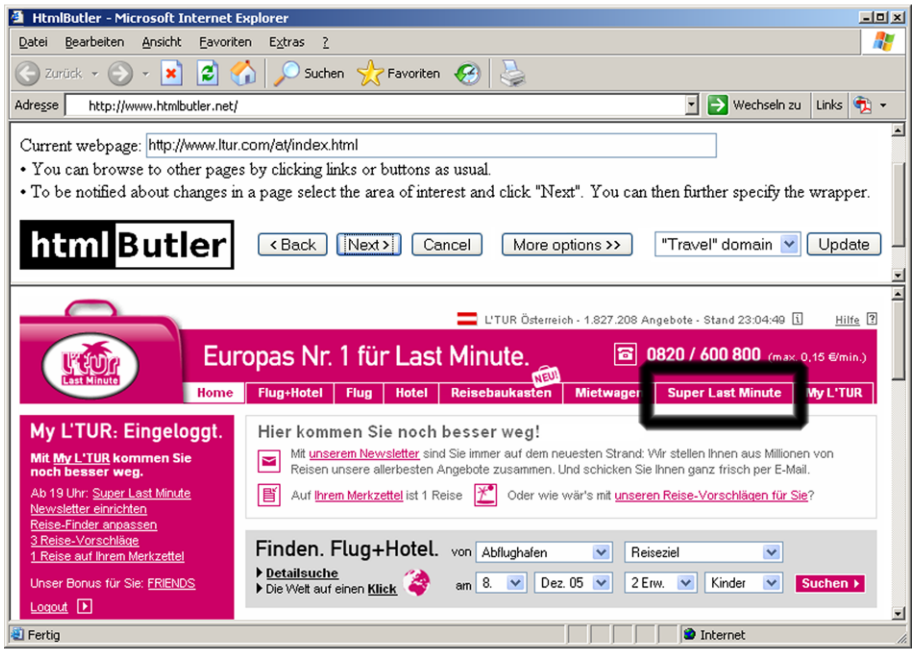

HtmlButler - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://www.htmlbutler.net/
Wechseln zu Links
testuser@htmlbutler.net | My Wrappers | My Account | Sign out
html Butler
Browse Wrappers Tutorial Popular Wrappers more »
http://www.ltur.at/
Submit Webpage Address or Keywords
©2005 htmlButler - Terms of Use - Privacy Policy - About htmlButler
Fertig
Internet

Figure 3.1: Start page to enter URL.



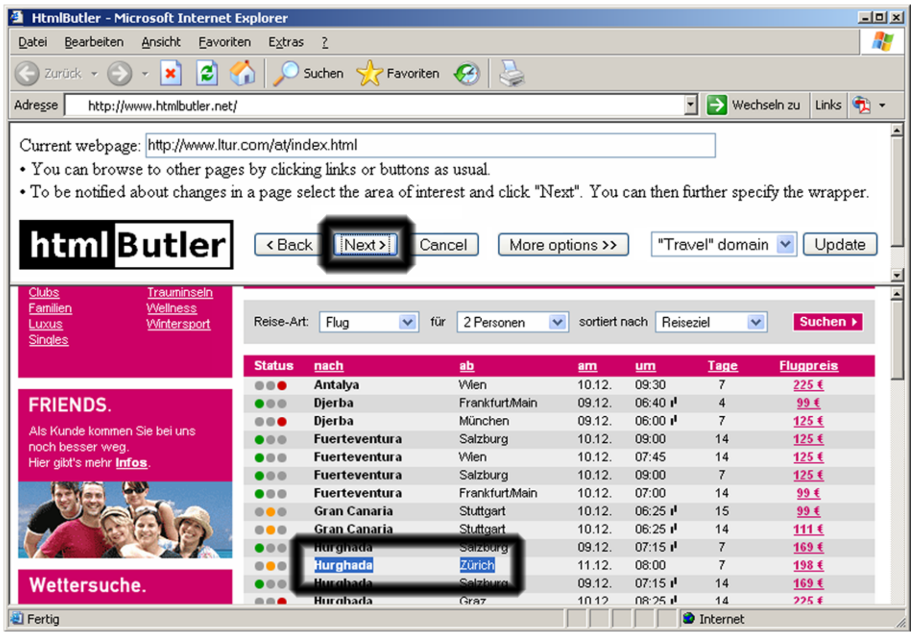
Current webpage: http://www.ltur.com/at/index.html
- You can browse to other pages by clicking links or buttons as usual.
- To be notified about changes in a page select the area of interest and click "Next". You can then further specify the wrapper.

My L'TUR: Eingeloggt.
Mit My L'TUR kommen Sie noch besser weg.
Ab 19 Uhr: Super Last Minute
Newsletter einrichten
Reise-Finder anpassen
3 Reise-Vorschläge
1 Reise auf Ihrem Merkzettel
Unser Bonus für Sie: FRIENDS
Logout

Hier kommen Sie noch besser weg!
Mit unserem Newsletter sind Sie immer auf dem neuesten Strand: Wir stellen Ihnen aus Millionen von Reisen unsere allerbesten Angebote zusammen. Und schicken Sie Ihnen ganz frisch per E-Mail.
Auf Ihrem Merkzettel ist 1 Reise
Oder wie wär's mit unseren Reise-Vorschlägen für Sie?

Figure 3.2: Navigate in lower frame.

-


HtmlButler - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://www.htmlbutler.net/ Wechseln zu Links
html Butler
< Back
Next >
Cancel
More options >>
"Travel" domain
Update
Clubs Familien Luxus Singles
Trauminseln Wellness Wintersport
FRIENDS.
Als Kunde kommen Sie bei uns noch besser weg. Hier gibt's mehr Infos.
Wettersuche.
Fertig
Internet

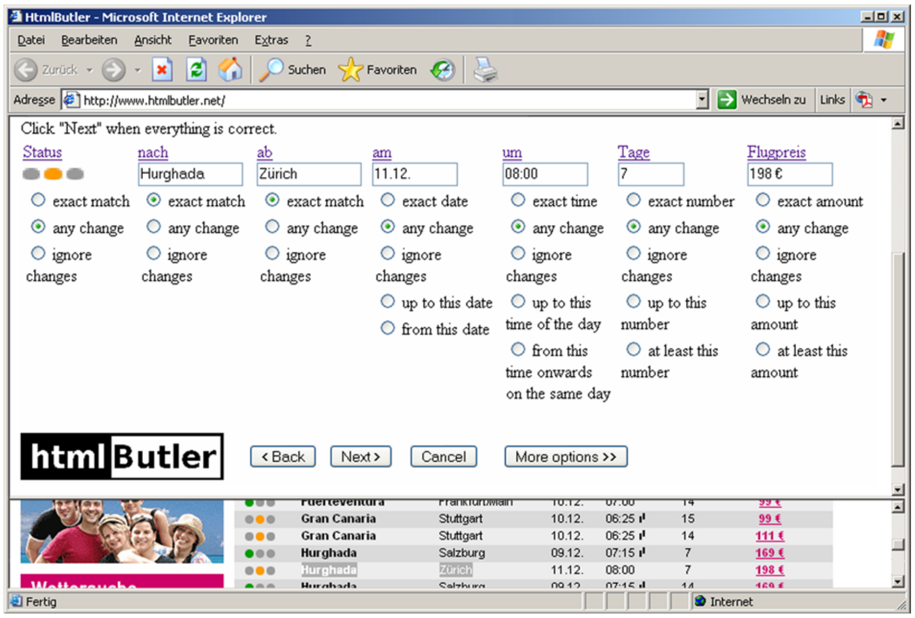
Current webpage: http://www.ltur.com/at/index.html

- You can browse to other pages by clicking links or buttons as usual.
- To be notified about changes in a page select the area of interest and click "Next". You can then further specify the wrapper.

Reise-Art: Flug für 2 Personen sortiert nach Reiseziel Suchen

Status	nach	ab	am	um	Tage	Flugpreis
●●●	Antalya	Wien	10.12.	09:30	7	225 €
●●●	Djerba	Frankfurt/Main	09.12.	06:40	4	99 €
●●●	Djerba	München	09.12.	06:00	7	125 €
●●●	Fuerteventura	Salzburg	10.12.	09:00	14	125 €
●●●	Fuerteventura	Wien	10.12.	07:45	14	125 €
●●●	Fuerteventura	Salzburg	10.12.	09:00	7	125 €
●●●	Fuerteventura	Frankfurt/Main	10.12.	07:00	14	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	15	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	14	111 €
●●●	Hurghada	Salzburg	09.12.	07:15	7	169 €
●●●	Hurghada	Zürich	11.12.	08:00	7	198 €
●●●	Hurghada	Salzburg	09.12.	07:15	14	169 €
●●●	Hurghada	Graz	10.12.	08:25	14	225 €

Figure 3.3: Select region of interest.

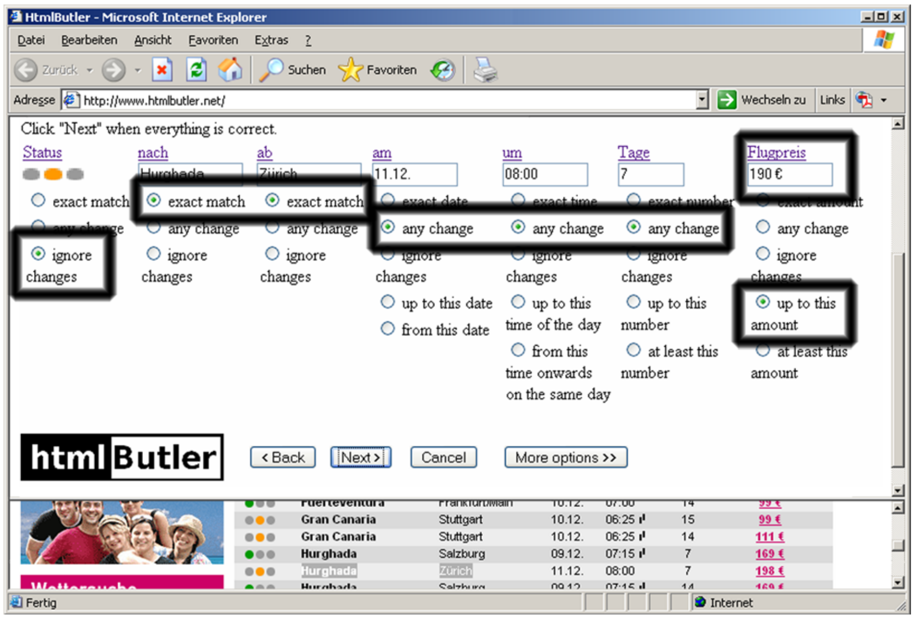

HtmlButler - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://www.htmlbutler.net/ Wechseln zu Links
html Butler
< Back
Next >
Cancel
More options >>
Wettersuche
Fertig
Internet

Click "Next" when everything is correct.

Status	nach	ab	am	um	Tage	Flugpreis
●●●	Hurghada	Zürich	11.12.	08:00	7	198 €
exact match	[✓ exact match]	[✓ exact match]	exact date	exact time	exact number	exact amount
[✓ any change]	any change	any change	[✓ any change]	[✓ any change]	[✓ any change]	[✓ any change]
ignore changes	ignore changes	ignore changes	ignore changes	ignore changes	ignore changes	ignore changes
			up to this date	up to this time of the day	up to this number	up to this amount
			from this date	from this time onwards on the same day	at least this number	at least this amount

Status	nach	ab	am	um	Tage	Flugpreis
●●●	Fuerteventura	Frankfurt/Main	10.12.	07:00	14	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	15	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	14	111 €
●●●	Hurghada	Salzburg	09.12.	07:15	7	169 €
●●●	Hurghada	Zürich	11.12.	08:00	7	198 €
●●●	Hurghada	Salzburg	09.12.	07:15	14	169 €

Figure 3.4: Possible options.

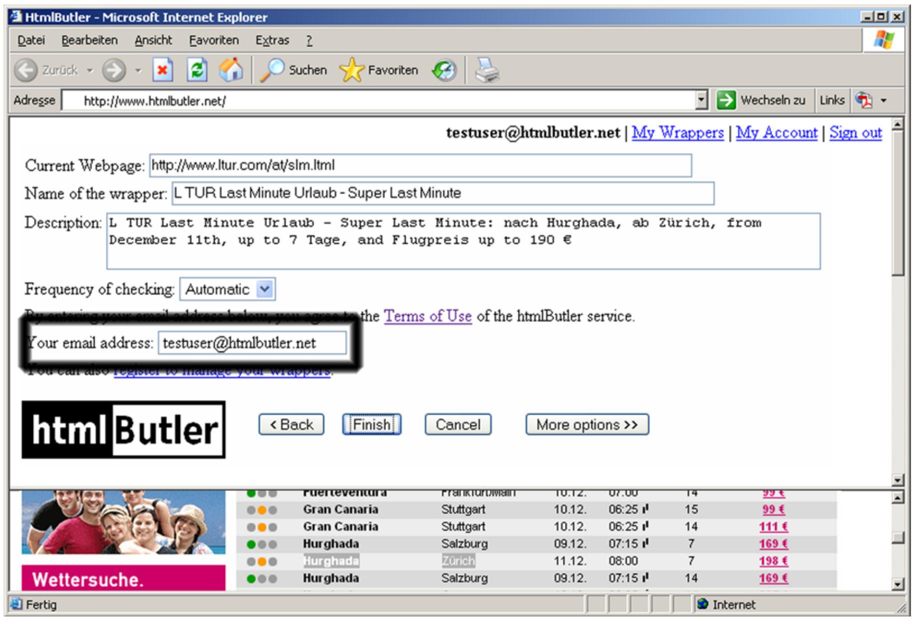

HtmlButler - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://www.htmlbutler.net/
Wechseln zu Links
htmlButler
< Back
Next >
Cancel
More options >>
Fertig
Internet

Click "Next" when everything is correct.

Status	nach	ab	am	um	Tage	Flugpreis
	Hurghada	Zürich	11.12.	08:00	7	190 €
exact match	[✓ exact match]	[✓ exact match]	exact date	exact time	exact number	exact amount
any change	any change	any change	[✓ any change]	[✓ any change]	[✓ any change]	any change
[✓ ignore changes]	ignore changes	ignore changes	ignore changes	ignore changes	ignore changes	ignore changes
			up to this date	up to this time of the day	up to this number	[✓ up to this amount]
			from this date	from this time onwards on the same day	at least this number	at least this amount

●●●	Fuerteventura	Frankfurt/Main	10.12.	07:00	14	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	15	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	14	111 €
●●●	Hurghada	Salzburg	09.12.	07:15	7	169 €
●●●	Hurghada	Zürich	11.12.	08:00	7	198 €
●●●	Hurghada	Salzburg	09.12.	07:15	14	169 €

Figure 3.5: Adjusted options.


HtmlButler - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://www.htmlbutler.net/
Wechseln zu Links
htmlButler
< Back
Finish
Cancel
More options >>
Wettersuche.
Fertig
Internet

testuser@htmlbutler.net | My Wrappers | My Account | Sign out

Current Webpage: http://www.ltur.com/at/slm.ltml

Name of the wrapper: L TUR Last Minute Urlaub - Super Last Minute

Description:
```
L TUR Last Minute Urlaub - Super Last Minute: nach Hurghada, ab Zürich, from
December 11th, up to 7 Tage, and Flugpreis up to 190 €
```

Frequency of checking: Automatic

By entering your email address below, you agree to the Terms of Use of the htmlButler service.

Your email address: testuser@htmlbutler.net

You can also register to manage your wrappers.

●●●	Fuerteventura	Frankfurt/Main	10.12.	07:00	14	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	15	99 €
●●●	Gran Canaria	Stuttgart	10.12.	06:25	14	111 €
●●●	Hurghada	Salzburg	09.12.	07:15	7	169 €
●●●	Hurghada	Zürich	11.12.	08:00	7	198 €
●●●	Hurghada	Salzburg	09.12.	07:15	14	169 €

Figure 3.6: Enter the email address.

3.1.2 Implementation

In the section above we described the interaction with the focus on the graphical user interface. Here the actions taking place simultaneously at the back-end of the system are described in detail. For wrapper creation the user runs through certain steps communicating with the htmlButler system. In Figure 3.7 five general steps and interaction loops are depicted. At the **START**-block, "the locator", the user enters the URL which he wants to visit to the htmlButler tool. Every HTTP-request is routed over the transparent web proxy, the answer of the server, usually a web page, is changed in a way that all links and form submissions are again routed over the htmlButler web proxy. The content of the page is analyzed and the page is assigned to a domain of knowledge (which already exists in the system). A domain of knowledge is represented by, e.g., hundred terms, which we call "profile of the domain". Also for any new page such a profile must be generated (hundred most relevant terms). This profile can then be compared to the profiles of all available knowledge domains. The page is assigned to the best fitting domain of knowledge. Relevant terms are determined by the use of TFIDF rating scheme. The similarity of two profiles can be determined by counting the common terms of two profiles. At block **I**, the "categorizer", the user can accept this content categorization through no additional action or change and update it. In any case the system stores the accepted category (domain of knowledge). This accepted category will be used for future assignments of exactly this page. In case of multiple users a simple majority vote takes place (category accepted by most users is chosen). When a certain threshold of users agrees with one categorization the htmlButler systems updates the profile of the domain of knowledge according to the page's profile. The system proceeds from block **I** to block **II**, the "analyzer", when the user selects and submits an area of interest in the current web page. The system loads the ontology corresponding to the previously determined category. The system analyzes the selection and tries to annotate parts of the selection using the concepts of the active ontology. This will be explained in more detail in section 3.1.3 below. At this stage, the user can do two things. If the user is not satisfied with the automatic annotations he can improve the system through changing them (see section 3.1.3 for details). If the user is satisfied with the annotations he proceeds with the wrapper configuration. The user can also define distinct thresholds for each token (annotated part of selection) to specify the wrapper's behavior in more detail (e.g., the user wants to be notified if an article's price drops below a certain amount). For every token the user can define, e.g., a maximum value. If the token changes and is above the given maximum value, the user wants to be notified. Similarly, the user can give a minimum or exact value that he is interested in. Typically, for numerical values a max/min/exact-match can be chosen whereas for string values only exact matches can be defined. The user submits this data to the htmlButler system and proceeds to block **III**, the "configurator", where he is just asked for notification details. If the user does not want to change further details he agrees and the system's state changes to the **END**-block, the "scheduler", where the actual wrapper is scheduled and an activation email is sent to the submitted email address. From each of the states, the user can restart the system again.

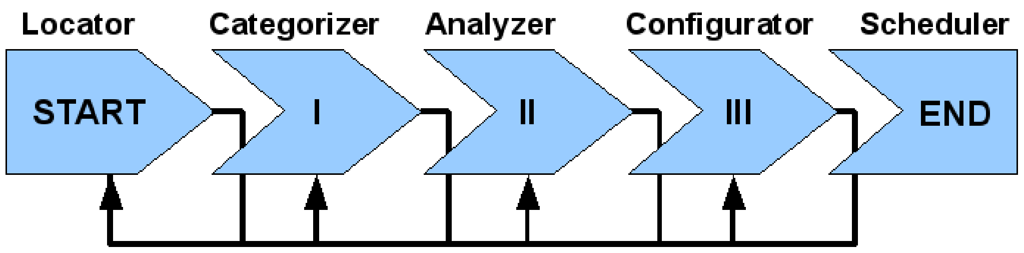


Figure 3.7: User interaction cycle in the htmlButler system.

3.1.3 The Role of Semantic Supported Wrapper Generation

Ontologies can enhance the process of wrapper creation. In terms of the user interaction cycle described above we are now at the stage where the domain of knowledge has been determined (block II, the analyzer) and the user has selected a part of the current web page in which he is particularly interested. In order to recognize the various concepts in the user's selection a number of patterns must exist for each concept in the ontology. Different patterns can be numbers, dates, time, amounts, tables, normal text etc. The user has now two possibilities for giving feedback to the system. In the simplest case the user just adds a string as pattern to a certain concept (by annotating this string with a concept). A more advanced user is able to add more complex patterns (e.g. nested regular expressions). In addition he also could add new concepts to the ontology. The ontology (hierarchy of concepts and corresponding patterns) is stored centrally on the htmlButler server. This means that whenever a user edits the ontology or the patterns corresponding to a concept the central and shared ontology is affected. Consequently, all users profit from the feedback given by any other user. On the other hand, multiple users make a voting process necessary. New patterns are introduced immediately. Patterns that are rated useless by a certain number of users are deactivated. Complete removal of a pattern is not possible in the htmlButler. A deactivated pattern is not deleted but is used as a kind of intelligent suggestion to the user when he is at the point of adding a new pattern. E.g., *London* is a deactivated pattern for the concept *location*. Therefore London is not automatically annotated in the user selection. Now if a user selects London and wants annotating it, the system suggests annotation with the concept *location*. This adds a positive vote for the pattern. Globally the decision is taken by a simple majority vote. The same is true for added and deactivated concepts and again complete removal is not possible. In the long run a maintenance process is necessary that truly removes unused concepts and patterns. The above described process is used to ensure practical quality. As shown in practice, *Collective Intelligence* already works for projects such as [71] and [72].

3.1.4 Server vs. Client-Server Solution

It was decided to provide two versions of the htmlButler project. The first was a plain server-side version, which had the drawback that there is an overhead in passing information between the client and the server at each step of the interaction. In the second version, parts of the user interaction, heuristics and tokenization functionality were moved from the server to the client side. This version had the advantage of less communication between server and client. With the plain server side solution, for instance, only the information about the user selection is prepared at the client-side and sent to the server where it is processed (the relative XPATH of the selected elements are created, the selection is tokenized, assumptions about the selected content are made and a feedback HTML page is generated and transferred back to the client browser). All these processing steps can be moved to the client, which decreases the data bouncing between server and client and dramatically improves the user interaction. The feedback loop, the direct access to the HTML page's document model and browser features like XUL [73] lead the following advantages of the plug-in version over the server-side version: (a) *Direct access to document model* - it is quite straightforward to get information about the selection and generating the XPATH for the selected entities. (b) *Improved Graphical User Interface* - through the use of XUL in the Mozilla [74] browser the user interface can be designed in a more sophisticated and user friendly way than only with plain HTML elements and JavaScript. (c) *Local data processing* - the selection the user made is tokenized, evaluated and processed locally and the user can change the suggested options without sending data to the server before finishing the wrapper creation. (d) *Alleviated recording of user action* - through the browser plug-in mechanism a sequences of user actions can be recorded if the user wants to create a wrapper which needs to authenticate itself to reach a certain web resource. With the client-server solution the disadvantage arises that the user must install a plug-in. There are multiple reasons why this may be inconvenient for a user. For instance, the user does not trust or is not allowed or able to install it or simply does not want to have yet another plug-in. Therefore the strategy is to provide the htmlButler service still as a simple server solution and for a better user experience through a browser plug-in.

3.1.5 Discussion

Using cooperatively created and shared ontologies in wrapper specification has been previously studied [75, 76, 77] and their approaches were tried to be combined. With the htmlButler, users could create a simple wrapper for an existing web page and be informed about certain changes. Although it was planned that users could be informed in various ways, e.g., email, SMS (short message service) over SMPP (short message peer-to-peer protocol), RSS-feeds [78], only the pull-style through a dedicated HTML-page and the email way were implemented. The htmlButler service allowed a user to be notified about changes in regions of interest on user-selected web pages observing a certain number of constraints, e.g., when a price of some product dropped below a certain value. It was planned for the purpose of keeping up an acceptable quality level in the shared ontologies to add "self correcting" features to the system. This feature was never implemented. Users should have been able to rate the usefulness of concepts and patterns contributed by other users. Furthermore it was the plan to use of group-ware tools such as WikiWiki-

Webs [79] which should facilitate free format organizational communication, allowing volunteer participation as in, e.g., the Wikipedia [71] project. In terms of requirements given for next generation Semantic-Web applications in [80] it was intended that the html-Butler system should have been made capable of a.) handling multiple ontologies, b.) use multiple user intelligence (Web 2.0 paradigm) and c.) being open with respect to non-semantic Web resources.

3.2 Project Analysis

There are many things that could possibly go wrong in a software development project [81]. In [82] symptoms and root causes of software development problems are listed and discussed in context with the Rational Unified Process as a solution. In [83] organizational, managerial, economic, technical and technological problems are identified and discussed as the root for failed projects. Generally one can distinguish between different problem sources such as a) human factors, b) technical, c) financial reasons, d) time issues, whereas the boundaries between these categories are blurred and combinations of kinds of these problems have again own names. The fact that things go wrong are empirically determined and published since 1994 in the annual “Chaos Report” [84] which are not seen without controversy in the research community [85, 86].
However, the major issues that arose during the project were not of technical nature, but rather concerned teamwork. A project retrospective (post mortem) as suggested in [87], was not possible since the different team members participated at disjunct times in the project. Nevertheless for this project a questionnaire (see Appendix B) was designed to gain insights into the developers’ view of the project progression and team experiences during their engagement in the htmlButler project.

3.2.1 Project Setup

The htmlButler project was funded by the Austrian FFG under the FIT-IT program line as a part of the NextWrap project and was carried out over a period of 24 month. It was not a pure academic project but in cooperation with a commercial company. There were one manager on the academic side, and two from the company. These shareholders were responsible for financial, technical, and scientific issues concerning the project. The development team had its office at the university whereas the cooperation partner was not located in the same town, but 200 km away which made frequent meetings and coordination of dates and times difficult. The academic manager of this project was full professor, so he was often occupied with different responsibilities and could not conduct daily meetings with the team. The initial team consisted of two PhD and one Master student. One PhD student was employed as a research assistant, one PhD student had a scholarship and the Master student participated with a student funding which was not included in the project budget. Every developer who worked in this project graduated with a Bachelor

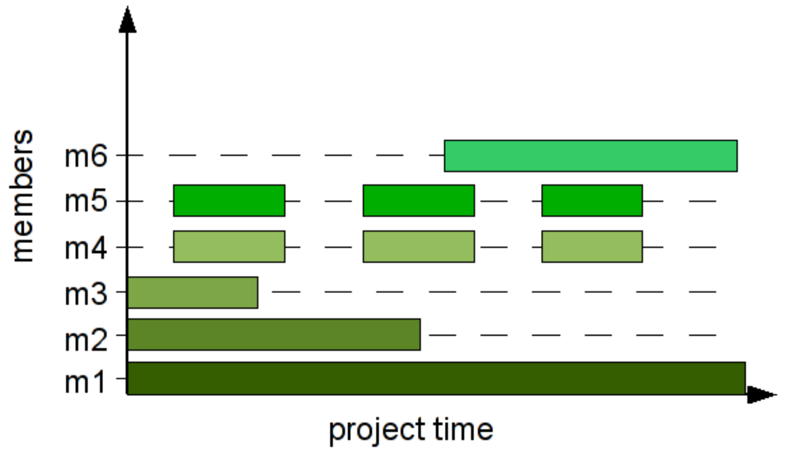


Figure 3.8: Member engagement during project time.

or Master degree in computer science and therefore had a general knowledge about agile software development methods and also software engineering. One of the developers had worked for some years as a professional full time programmer in the Austrian IT-industry, another had some practical work experiences through project work with a software company. All in all there where 6 people involved in the htmlButler project, where only one participated the full 24 months, and one three months of the project period. The others had partial engagements from 6 to 12 months (see Figure 3.8). During their engagement three members worked full-time (40 hours per week) and three part-time (10 to 20 hours per week) in the project. At the project start it was agreed to develop using the XP methodology and use this method as strict as possible. The feedback how this was realized is described in Section 3.2.5.
Because most of the developers were not involved during the whole project period but only partly at the beginning, at the end, or on demand during the project, the collected data are sometimes contradictory. For instance when questions about the project beginning or end were asked, it must be kept in mind that "the beginning" and "the end" of the project was individually different.

3.2.2 Questionnaire Design

The htmlButler post mortem questionnaire (Appendix B) consisted of the following 6 sections a.) general information, b.) teamwork, c.) management, d.) XP practices in general e.) experience with XP practices and f.) fundamental project questions. The goal was to find out how the developers experienced the htmlButler project in general, concerning the teamwork, the management, and especially about the XP practices which were intended to be applied as the development method of choice. Furthermore the questionnaire was conducted to draw conclusions about the problems in the project and how things could be done in a different way to improve the outcome of similar projects in future. The section

“fundamental project questions” of the questionnaire was inspired by the “Chaos Report” which suggests a method to estimate the “success potential” of a project by means of ten question categories, each with 5 closed questions. The goal was to check whether the developers’ intuitive project appraisal could be confirmed by this method.
For the sake of anonymity the distinct votes are not published since it would be possible to draw conclusions from the data and correlate them with the team members.

3.2.3 Team and Teamwork

Results In Table 3.1 the general questions to the role of the developers in the project and the project itself are summarized. These questions could be answered on a scale between 0 and 5 where 0 meant “poor” and the value 5 meant “excellent”. To the question if the participants enjoyed their time in the project only one stated that the time was not enjoyable. The rest voted between 2 and 4, which shows all in all that this project was not a total catastrophe in human aspects. The average vote as well as the median “enjoyment” was above the average. The area of work in the project as well as the project target was also clear to the developers (above the average) although the minimum vote was quite low which shows that there was definitely a problem in the team. The perceived clarity of the project target of the project management and upper management was rather low indicating a problem in the information flow from management to the developers since the median was 2. The motivation at the start of the project was equally high amongst the development, between 4 and 5 meaning almost excellent. Towards the end of the project the motivation had decreased, with an average value of 2 for “Motivation at the end of your time in the project”.
Questioned about the number of developers for the project in terms of too many or too few on a scale from 0 to 5 where 0 meant too few and 5 meant too many, the outcome was that the average was 2.2 and the median was 2 which indicates a slight tendency that there were too few people on the project (Table 3.2). The number of co-developers in the project varied between 0 and 4 which shows a dynamic team size and that at no time in the project all participants worked together (table 3.3). The developer’s self perception for ability to team work was clearly positive with an average of 4.3 on a scale between 0 and 5 where 0 meant poor and 5 meant excellent (table 3.4). Similarly the self perception of the ability to independently work on a problem was between 2 and 5 and 3.3 in average whereas the co-developers’ ability was rated between 2 and 4 but only 2.6 in average. This discrepancy shows that the team was not able to work well together and did not act as a team therefore could not grow to their full potential. Since 50% of the members have worked before in team projects it cannot simply be lead back to team inexperience. To the question if team members hat problems with another member 50% answered with “yes” which supports previous conclusion that the team could not perfectly work together. One member stated that the problems were based on ”misunderstandings”. A second stated that it was due to the “arrogance, ignorance and incompetence” of the member he or she had problems with. The team spirit at the start of the member’s engagement in the project was rated high by all members between 3 and 4 leading to a value of 3.7 in average. The

team spirit at the end of the member's engagement in the project was lower (Table 3.4) between 0 and 3 with an average of 1.2. The distribution of work between the members was perceived as fair. Only one member stated it was not fair distributed but explicitly mentioned that he or she did not to work more than others. A majority of 83% perceived the level of knowledge as different between team members and the knowledge transfer as nonexistent. Only one team member perceived the level of knowledge as equal between team members (Table 3.5). Concerning developer meetings, the frequency of internal and meetings with shareholders were rated 2.2 and 2.7 in average 2.5 and 3 in median (table 3.6) on a scale of 0 meaning "too few" and 5 "too many" indicating a "right" amount of meetings. The usefulness of the project meetings was rated 2.5 for the internal and 2.2 for the meetings with the shareholder indicating a neutral usefulness. Although the participants said they had no problems to make themselves understood in the meetings (Table 3.5) there must have been communication problems rendering the meetings not as useful as necessary for agile development (e.g., see "seven habits of effective pair programming" [88]). As a conclusion the participants were asked whether they would participate in this project again. One person (16.7%) answered with "yes" but the majority of 5 (83.3%) denied.

Discussion It can be seen as a disadvantage that there was not a stable group with similar skills, payment and contracts from the beginning. The team never really left the storming phase [89] through the dynamic setup, and hence never really became productive. The fact that the motivation decreased towards the end of the project can be ascribed to the heterogeneous team, infrequent participation in the project by some team members, dynamic changes of the project scope, as well as the discrepancy between project targets the management communicated at different times. One vote for the "end motivation" was "excellent" which was an outlier, since despite one vote with 5 points, the average still only reached 2. If one considers that the engagements were not over the whole project and distributed over the project (Figure 3.8) it is interesting that the team spirit was perceived as low individually by each member.
Many problems experienced by the team could be attributed to problematic pair line-ups, e.g., expert-novice, introvert-introvert pairing as well as "my partner is a total loser" and other excess ego problems, as have been described in [88].

Question (0..“poor” and 5..“excellent”)	avg.	med.	max.	min.
enjoyed time in project	2.8	3.5	4	0
area of work in the project clear	3	3.5	4	0
project-target clear to dev.	3	3.5	4	1
project-target clear to the project- and upper managers	2.5	2	4	1
motivation at the START of your time in the project	4.5	4.5	5	4
motivation at the END of your time in the project	2	2	5	0
own contribution to the project	3.5	4	4	2
own ability to work independently	4.3	4.5	5	3
rate your payment	2.2	2	4	2

Table 3.1: General project- and self-perception of the team members.

Question (0..“too few” to 5..“too many”)	avg.	med.	max.	min.
Development power during the project	2.2	2	3	1

Table 3.2: Perceived development power during the project.

Question	avg.	med.	max.	min.
Number of co-developers during project time	2.8	3	4	0

Table 3.3: Number of co-developers in the team.

Question (0..“poor” to 5..“excellent”)	avg.	med.	max.	min.
Self perception of teamwork ability	4.3	4	5	4
Co-developers’ teamwork ability	2.8	3	4	2
Self perception to work independently	3.3	3	5	2
Co-developers’ ability to work independently	2.6	3	4	2
Team spirit (moral/mood) - START of project	3.7	4	4	3
Team spirit (moral/mood) - END of the project	1.2	2	3	0

Table 3.4: Self-, team- and fellow team member perception.

Question	yes	no
Was the work-distribution fair among the team members	5	1
Did you have to work more than other team members	0	6
Did you work in a project team before	3	3
Did the team members have equal knowledge	1	5
... if no - did the knowledge transfer work	0	5
Did you have problems making yourself understood	0	6
With the experience made, would you participate again?	1	5

Table 3.5: General questions concerning team and teamwork.

Question (0..“too few” to 5..“too many”)	avg.	med.	max.	min.
Frequency of internal meetings	2.2	2.5	3	0
Frequency of the general project meetings (shareholder)	2.7	3	4	1

Table 3.6: Perceived frequency of meetings.

Question (0..“waste of time” and 5..“a must”)	avg.	med.	max.	min.
Usefulness internal meetings	2.5	3	4	0
Usefulness of the general project meetings (shareholders)	2.2	3	4	0

Table 3.7: Perceived usefulness of meetings.

3.2.4 Management

Results The developers perceived a different number of leading managers, between 1 and 3 managers. They had contact with between one and three managers (more than two managers in average). The agreement level among the managers was perceived as “moderate” and was rated between 0 and 5, similar to the perceived management skills which were between 0 and 4. The leadership style of the project manager on a scale between 0 to 5 (“weak” to “dominant”) was rated 1.5 in average meaning rather “weak”, whereas the project organization was rated 1.2 in average on a scale between 0 to 5 (“chaotic” to “plan driven”). The overall satisfaction with the project managers and the shareholders were 2 in average. Furthermore, 50% (3) of the development team members perceived no clear hierarchy in the management and 66% (4) of the members stated that only a poor match between the objectives of the project mangers existed. 83% (5) thought that no concrete project plan existed at the start of the project. During the project this value was at 66% (4) and at the end of the project the developers again 83% (5) stated, that there was no common project plan. 66% of the team members stated that in hindsight they thought that the project could have been successfully accomplished under different circumstances.

Discussion First, it can be observed that the management structure of the project was not completely clear to the developers. This can be explained with a high turnover of people in the team and consequently sparse contact between single developers and the management. Some team members did not even meet all the involved managers or know about them. This could potentially have been solved by regular meetings with the shareholders during the whole project. The low overall satisfaction of developers with the management indicates a strained relationship between developers and the management. Again, this might be related to the loose contact to the management but also to the frequent changes of objectives. The latter point is also indicated by the high number of developers who perceived no common project plan (at the start, during the project and at the end). The low ratings in the section “management” correlate with the low ratings in the section “team and team work”. According to the answers to the questions in the section “management”, the team members thought that most of the problems could have been prevented through more contact to the management, a clearer hierarchy and a stricter leadership at the beginning of the project.

Question (0..“poor” and 5..“excellent”)	avg.	med.	max.	min.
How many managers were involved in the project	2.3	2.5	3	1
With how many managers did you have contact	2.3	2.5	3	1
Agreement among the managers	3	3.5	5	0
Perceived management skills of the managers	2.7	3	4	0
Leadership style of the project manager (0...“weak” to 5...“dominant”)	1.5	1.5	3	0
Organization of the project (0...“chaotic” to 5...“plan-driven”)	1.2	2	2	0
Satisfaction with the project managers’ work during the project	2	1.5	5	0
Satisfaction with the shareholders	2	2.5	3	0

Table 3.8: Perceived management qualities.

Question	yes	no
Was there a clear hierarchy in the management	3	3
Did the targets of the managers match	2	4
Do you think there was a common project plan at the start of the project	1	5
Do you think there was a common project plan during the project	2	4
Do you think there was a common project plan for the end phase of the project	1	5
With your knowledge today do you think this project could be done successfully?	4	2

Table 3.9: General questions concerning the management.

3.2.5 XP Practices

Results In Table 3.11, general questions concerning the own experience level and the project are described. The scale was between 0 = "poor", 1 = "below average", 2= "average", 3 = "good", 4 = "very good" and 5 = "excellent".
33% of the developers (2 people) had worked with agile methods before in practice. The same percentage had practiced pair programming before entering the project. 83% of the developers had a positive attitude towards agile development methods (Table 3.10). Developers rated their own experience with agile methods as "average", with a tendency towards "good". The experience level in programming was rated "good" in average and "very good" as a median.
The developers rated the XP methodology to be appropriate for the project with a tendency from "good" to "very good". The XP methodology was rated to be "average" to "good" suited for this team, whereas the enforcement of the practices was rated "average" with a tendency to "below average". Questions concerning the application of XP practices in the htmlButler project used a scale from zero to 10, with zero being the worst rating (see Table 3.12 for the detailed verbalization of the scale). In Table 3.13 the average, median, maximum and minimum values of the ratings concerning the application of XP practices in the htmlButler project are described. Six practices, were enforced above the average including a.) "coding standards" with 6.5 in average and 8.5 in median, b.) "collective code ownership" with 7 in average and 8 in median, c.) "refactoring" with 6.2 in average and 7 in median, d.) "simple design" with 5.8 in average and 6.5 in median, e.) "growth" (team getting smarter with the time) with 5.5 in average and 6 in median and finally f.) "automated unit tests" with 5 in average and 5.5 in median. All other practices were not enforced and the least enforced practice was pair programming.
In Tables 3.14 and 3.15 XP practices are rated whether they are a.) helpful, b.) enjoyable c.) widely used, d.) easy to learn, e.) easy to apply and finally f.) easy to introduce in a team. The scale to rate these practices ranged from 0 = "strongly disagree" to 5 = "totally true". The top five of helpful practices were a.) "automated unit testing", together with , b.) "coding standards" with a rating of 5 in average, c.) "simple design" with 4.4 in average, and d.) "pair-programming", together with e.) "continuous integration", both rated with 4.3 for helpfulness. The rating, whether these practices are enjoyable were different. The top five were a.) "simple design", together with b.) "whole team" with an average of 4.4, c.) "automated unit testing", together with d.) "pair programming" with an average of 4 and finally e.) "continuous integration" with an average rate of 3.8. It is interesting that the practices "simple design", "automated unit testing", "continuous integration" and "pair programming" are considered as useful and enjoyable by the team. Asked whether these practices are widely used, another top five have resulted a.) "coding standards" with an average of 3.8, b.) "refactoring" with 3.6, c.) "sustainable pace" with 3.5 , d.) "metaphor" with 3.3 and e.) "continuous integration" with 3 in average. The category "easy to learn" is lead by the following top five practices: a.) "daily standup" with 5 in average, b.) "coding standards", together with c.) "collective code ownership" with 4.2 in average, d.) "sustainable pace" with an average of 3.8 and finally e.) "small releases" with

3.6 in average. The top five practices rated as “easy to apply” are a.) “sustainable pace” with 4.6 in average, b.) “daily standup”, together with c.) “continuous integration” with 4 in average, d.) “coding standards” with 3.8 and finally e.) “small releases” with 3.2 in average. The top five practices rated as “easy to introduce” are a.) “sustainable pace” with an average of 4.6, b.) “continuous integration” with 4.3 in average, c.) “metaphor” with 3.3 and finally d.) “coding standards”, together with e.) “refactoring” with 2.7 in average. The practice of “coding standards” is the only one which appears in all categories in the top five but the category “enjoyable”. The practice “continuous integration” appears in all the top 5 of the categories but the category “easy to learn”. The core practice of XP, “pair programming” is considered to be “helpful” and “enjoyable” but the hardest to introduce in a team with an average value of 1.

Discussion It is interesting to see that all XP practices are rated above the middle of the scale, which shows that these practices are widely recognized as useful. Whatever process is defined, if the involved people do not know how to work together, the process will not be followed [19]. This was experienced in the htmlButler project, where it can be seen that the XP practices were not really enforced above average. Continuous integration is the only practice which is considered as helpful, enjoyable and widely used. The data show that pair programming had not been enforced although it was rated as helpful in general. A possible explanation for this apparent contradiction might be that the difficult team setting led developers to silently ignore this practice. Furthermore, the data give a hint that there were possibly at least one “very motivated” and one very “disappointed” member in the team due to the maximum and minimum votes.
The author of this thesis has personally observed during the htmlButler project, that in the beginning it was intended to adhere to all XP rules. The team started to use different web based tracking tools and a spreadsheet application. However it soon turned out that this was too heavy-weight for this small team so it was switched back to analog and the whiteboard was used to track the progress. The whiteboard was also neglected, since at the beginning of the project the team did not have a clear vision and needed to collect ideas and do some prototyping in different fields. It became obvious that the knowledge about the used programming language was not homogenous (see results in Section 3.2.3 concerning levels of knowledge). Some of the pairing problems described in [88] became reality. This rendered pair programming almost impossible for this team. This personal assessment correlates with the fact that only two people rated pair programming to have worked in the team.

Question	yes	no
Do you favor agile methods over traditionally software development methods	5	1
Have you worked in a project utilizing agile methods before	2	4
Did pair programming work in your team	2	4
Did the team follow the test first paradigm	3	3
Did the team follow the collective code ownership	5	1

Table 3.10: General questions concerning agile methods in the htmlButler project - part 1.

Question (scale between 0..poor and 5..excellent)	avg.	med.	max.	min.
Own general experience level with agile methods	2.7	3	4	0
Own general experience level in programming	3.3	4	4	1
Was the XP paradigm appropriate for the intended project	3.2	4	4	1
Was the XP paradigm appropriate for the team	2.5	3	4	0
Rate the enforcement of the established agile methods	1.8	2	3	0

Table 3.11: General questions concerning agile methods in the htmlButler project - part 2.

Effectivity level	value
Fanatic	10
Always	9
Regular	8
Often	7
Usually	6
Half and Half	5
Common	4
Sometimes	3
Rarely	2
Hardly ever	1
Disagree/Never	0

Table 3.12: Scale for possible answers for questions in Table 3.13.

XP practice (scale see table 3.12)	avg.	med.	max.	min.
Coding Standards	6.8	8.5	9	1
Collective Ownership	7	8	9	1
Refactoring	6.2	7	9	0
Simple Design	5.8	6.5	9	1
Growth (team getting smarter over time)	5.5	6	8	1
Automated Unit Tests	5	5.5	9	1
System Metaphor	4.8	5	8	1
Continuous Integration	4.5	5	8	1
Lessons Learned	4.5	5	7	0
Morale	4.2	5	7	0
Short Releases	4.8	4.5	9	0
Sustainable Pace	4	4.5	6	0
Test First Design	4.8	4	8	2
Synergy	4	4	8	1
Stand Up Meeting	3.5	4	7	0
Artifact Reduction (do 'just enough' documentation)	4	3.5	6	2
Release Planning	3.3	3.5	6	1
Customer Access	3.3	3	8	0
Customer Acceptance Tests	3	2	9	0
Pair Programming	2	2	5	0

Table 3.13: Application of XP practices in the htmlButler project.

Experienced XP practice		helpful	enjoyable	widely used	easy to learn	easy to apply	easy to introduce
Planning Game	avg.	3	2.6	1	2	2.8	2
	max.	5	4	2	3	4	3
	min.	1	1	0	1	1	1
Small Releases	avg.	3.8	3.4	2.2	3.6	3.2	2.5
	max.	5	5	5	5	5	5
	min.	1	1	0	1	1	1
Metaphor	avg.	3.5	3.3	3.3	3	2.8	3.3
	max.	4	5	4	4	4	4
	min.	3	2	2	2	2	3
Simple Design	avg.	4.4	4.4	2.6	2	2.6	2.6
	max.	5	5	4	3	3	3
	min.	3	4	2	1	2	2
Testing	avg.	5	4	2.5	3	2.7	2.3
	max.	5	5	3	4	4	3
	min.	5	3	1	2	2	1
Refactoring	avg.	4.2	3.2	3.6	2.2	3	2.7
	max.	5	5	4	3	4	3
	min.	3	2	3	2	2	2
Pair Programming	avg.	4.3	4	1.7	3.3	3	1
	max.	5	5	3	5	5	2
	min.	3	3	0	1	1	0
Collective Ownership	avg.	4.2	3	2	4.2	2	2
	max.	5	5	3	5	2	2
	min.	3	2	1	3	2	2
Continuous Integration	avg.	4.3	3.8	3	3.5	4	4.3
	max.	5	5	4	5	5	5
	min.	3	2	2	2	3	3
Sustainable Pace	avg.	2.6	3.4	3.5	3.8	4.6	4.6
	max.	3	5	4	5	5	5
	min.	2	2	3	3	4	4
Scale between 0..“strongly disagree” to 5..“totally true”							

Table 3.14: Personal experienced XP practices - part 1.

Experienced XP practice		helpful	enjoyable	widely used	easy to learn	easy to apply	easy to introduce
Coding Standards	avg.	5	3.4	3.8	4.2	3.8	2.7
	max.	5	5	5	5	5	3
	min.	5	3	2	3	2	2
Daily Standup	avg.	4	2.7	2.7	5	4	2.5
	max.	5	4	3	5	5	3
	min.	2	1	2	5	2	2
Whole Team	avg.	4	4.4	2.2	3	2.4	2
	max.	5	5	3	4	3	2
	min.	3	4	1	2	2	2
Scale between 0..“strongly disagree” to 5..“totally true”							

Table 3.15: Personal experienced XP practices - part 2.

3.2.6 Fundamental Project Questions

This part of the questionnaire was inspired by the "Chaos Report" simply to find out whether the intuitive estimation of the project success correlates with the "success potential" computed with the algorithm the "Chaos Report" from 1994 suggests. In the "Chaos Report" ten categories for potential project success were determined each with a different relevance. Of course, these categorizations are neither up to date (were revised in later reports) nor scientifically beyond doubt, but they can give a hint to the potential for success of a project. The "Chaos Ten" were weighted as follows a.) user involvement with 19%, b.) executive support with 16%, c.) clear requirements with 15%, d.) proper planning 11%, e.) realistic expectations with 10%, f.) small project milestones with 9%, g.) competent staff with 8%, h.) ownership with 6%, i.) clear vision and objectives with 3% and finally j.) hard working focused staff with 3%. Each category consists of 5 questions which can be answered either with "yes" or "no". According to the question's category the percentage of a question is computed with the formula: $\frac{\#yes*Qweight}{\#members}$. This formula yields a percentage. $\#yes$ represents the number of positive answers for this question, $Qweight$ represents the weight for this question according to its category, and $\#members$ the number of people who took part in the survey. $Qweight$ is chosen such that, if all answers were "yes" the sum over all categories would be 100%. The actual sum over the percentages of all categories gives the success potential of this project. If this factor is below 35%, it is better to keep away from this project. If it is larger than 35% but lower than 70% the project is categorized as a "high risk" or "research" project. If the factor is larger than 70%, the project has a good chance of success. In the questionnaire, conducted for the htmlButler project, for this analysis, two values for each question were given one for the project start, and one for the project end.

Results Given the answers of the developers to the questions of the section "fundamental questions" in the questionnaire, the computed confidence in the project success was a little less than 40%, using the data for the beginning of the project. Using the data for the end of the project, the computed confidence in the project success was a little more than 34%.

Discussion Since the questionnaire was answered after the project, the values are not accurate since the participating people were not all involved in the project at the same time. However all estimations clearly marked the htmlButler project a "high risk" project at the border to a project which will likely become a failure. This outcome correlates with the overall low ratings in other parts of the questionnaire. At the same time this outcome also shows that it would have been possible to create an awareness of the "high risk" situation already at the beginning of the htmlButler project. It is possible that such awareness would have helped to conduct the project differently and potentially more successfully.

User Involvement - 19%	project phase	yes	no
Do we have the right users?	start end	3 3	3 3
Users involved often and from the beginning?	start end	1 1	5 5
Quality recording of users?	start end	2 1	4 4
Is user involvement encouraged?	start end	2 2	3 3
Are the needs of the users known?	start end	4 4	2 1

Table 3.16: Fundamental questions concerning user involvement.

Management Support - 16%	project phase	yes	no
Do we have the key executives?	start end	1 1	5 4
Have the key executives financial share in the project?	start end	2 2	3 3
Is a failure of project acceptable?	start end	3 3	2 2
Is there a well defined project plan?	start end	1 1	5 4
Have the team members financial share in the project?	start end	1 1	5 4

Table 3.17: Fundamental questions concerning executive management support.

Clear Statement to Requirements - 15%	project phase	yes	no
Do we have a clear vision?	start end	3 3	3 3
Do we have functional analysis?	start end	3 3	3 3
Do we have risk analysis?	start end	0 0	6 6
Do we have a business case?	start end	4 3	2 3
Is the project progress measurable?	start end	4 5	2 1

Table 3.18: Fundamental questions concerning a clear statement to requirements.

Proper planning - 11%	project phase	yes	no
Do we have a problem statement?	start end	3 3	2 2
Do we have a solution statement?	start end	2 2	3 3
Do we have the appropriate people in the team?	start end	2 2	3 3
Do we have a well-founded hard specification?	start end	0 0	5 0
Do we have achievable milestones?	start end	4 2	1 3

Table 3.19: Fundamental questions concerning proper planning.

Realistic Expectations- 10%	project phase	yes	no
Do we have the expectations clearly specified?	start end	2 1	3 4
Do we have prioritized requirements?	start end	3 2	3 3
Do we have small milestones?	start end	5 2	1 3
Are we able to react well to changes?	start end	3 2	3 3
Are we able to practice prototyping?	start end	4 3	2 2

Table 3.20: Fundamental questions concerning realistic expectations.

Small Project Milestones - 9%	project phase	yes	no
Do we apply the 80/20 rule?	start end	1 0	3 4
Do we follow a top-down design?	start end	2 1	4 4
Do we have time limits?	start end	4 4	2 1
Do we use a prototyping tool?	start end	2 2	3 3
Can we measure the progress?	start end	4 4	2 1

Table 3.21: Fundamental questions concerning small project milestones.

Competent Staff - 8%	project phase	yes	no
Do we know which skills are needed?	start end	5 5	1 1
Do we have the right people in the team?	start end	2 2	4 4
Do we have a training program?	start end	0 0	6 6
Do we have incentives (awards, bonuses)?	start end	0 0	6 6
Does the staff have an overview?	start end	5 5	1 1

Table 3.22: Fundamental questions concerning competent staff.

Ownership - 6%	project phase	yes	no
Do we have well-defined roles?	start end	2 1	4 4
Do we have a well-defined organization?	start end	0 0	6 5
Do we know our own roles?	start end	3 1	3 4
are there success awards/bonuses offered?	start end	0 0	5 5
Is everybody committed to the project?	start end	4 3	2 3

Table 3.23: Fundamental questions concerning ownership, clear roles and tasks.

Clear Visions and Objectives - 3%	project phase	yes	no
Do we all share the same vision?	start end	3 1	2 4
Does the vision fit the company objectives?	start end	4 2	1 3
Are the objectives achievable?	start end	2 2	3 3
Are the objectives measureable?	start end	4 3	1 2
Do we check honestly for meaningfulness?	start end	1 0	4 5

Table 3.24: Fundamental questions concerning clear visions and objectives.

Hard working focused staff - 3%	project phase	yes	no
Is there appeal for success?	start end	4 2	1 3
Do we concentrate on quantifiable deliverables?	start end	2 2	3 3
Do the members all have part-ownership?	start end	2 2	3 3
Do the members work well together?	start end	1 2	4 3
Do the members trust each other?	start end	2 3	3 2

Table 3.25: Fundamental questions concerning the staff.

3.3 Conclusion

The project analysis of the htmlButler project shows, that the project did not went well and that there were severe shortcomings in implementing the XP methodology beside massive team problems. The group of developers was inhomogeneous concerning age, culture, background and skills and therefore was too instable to grow to a team or into a stage of performance. Two people left during the first 12 months, one entered the team after 12 months, and two developers, who were mainly concerned with usability research, worked only short phases in the project. All ratings show one outlier with a very low value for all questions. This indicates that one team member was particularly disappointed with all aspects of the project. The relation between development and management was suboptimal. The managers of the commercial partner were partially not known by the developers, due to infrequent meetings. The local manager of the project was too occupied with other projects and duties that the contact to the development team was too loose. This too was a reason for the perceived lack of management qualities. Additionally, the developers had the impression that between the academic manager and the cooperation partners the objectives were not in sync. Only one of the 6 involved developers would participate in the same project again. In the textual feedback given within the post mortem questionnaire, the developers mentioned lack of project focus, lack of management support as well as lack of strict roles, hierarchy and leadership, besides the suboptimal team staffing, as the main reasons for the mediocre outcome of this project.
The goals of the htmlButler were very ambitious. In addition, it was scheduled with a limited financial and time budget. To be able to create a meaningful web-based wrapper tool with decent usability, following topics have to be covered: a.) information integration which is concerned with machine learning techniques for extracting information from Web sources, b.) semantic modeling, creation of semantic models of the wrapped sources for integration and combination with other sources, c.) record linkage to align data across sources, and d.) data integration and plan execution to automatically integrate data from the Web. These topics represent a large field of research and a small team of only one or two PhD students cannot even come close to covering it. For instance, the information integration research group (IIRG) around Professor Craig A. Knoblock[2] from the Information Sciences Institute, Department of Computer Science of the University of Southern California, has published 80 papers with over 40 different involved scientists between 1992 and 2005 on the above-mentioned fields. This means, that when the htmlButler project started, this group alone had a headstart of 14 years of research.
The analysis of the htmlButler project further shows that it would have been possible to become earlier aware of project problems, through the application of project success probability estimations, e.g., the "Chaos Report" algorithm, as well as an assessment of the field of research and its groups, e.g., the IIRG. This would have made it possible to earlier react to a large part of the problems which occurred during the htmlButler project. Consequently this could have made it possible to increase the success of the htmlButler project.

[2]Information integration research group: http://www.isi.edu/integration, visited February 2007.

Chapter 4

Agile Methods in Austrian IT-Industry - Results of an Empirical Survey

This chapter is based on the paper [57] already published in 2008 where the results of the empirical study concerning agile software development methods and practices used in the Austrian IT-Industry, was analyzed. The main target was to gather information about the use of agile software development methods in general and particularly with regard to the agile practice of pair-programming. The author's experiences in the commercial software industry as well as in academic projects (see Chapter 3) raised the question of how agile practices are implemented in practice. Data was collected which of the most simple to adopt agile practices and tools are used in practice and what are the future plans to introduce or enforce the use of agile software development methods in general or the pair-programming practice specifically. Another aim was to determine the perceived adherence to delivery dates in the software development industry in general. The author tried to reach two people of each company to make out the difference between management and development on the questioned topics. The base of the questions which were asked in the telephone interviews can be read in Appendix C.

4.1 Introduction

Austria's software industry is a strongly expanding branch of business [90]. This is still a fact in the year 2008 [91] where the growth of this business branch is estimated to be between 5.4% and 6% until the year 2011 [92]. 2002 the results of an empirical study among small and very small enterprises in Austria were published concerning the used processes, techniques, and tools as well as problems during software development [93]. In this paper a similar empirical study is conducted but the main focus here is to find out the use of agile software development methods in general and the agile practice of pair-programming (PP) in particular. Furthermore, it is not limited to small and very small enterprises and is of more recent data. The survey was designed to find out information about:

a) the general size of the company and the average development team size,
b) whether there is knowledge or at least awareness of agile software development methods and whether there is a desire for further information concerning this topic,
c) the current use of agile software development methods as well as the use of certain practices which are considered to be implementable with a low amount of effort [58, 20] (such as the test-first[1] approach or continuous integration, e.g., Hudson [94] and Cruisecontrol [95]) as well as the reasons for no adoption of agile methods (AM),
d) the experience with agile methodologies and pair-programming in practice, the intention to apply agile methods or practices in the future in the company's development process, and finally,
e) the general spirit of adherence to delivery dates in the role of a client (outsourcing) as well as contractor of software products. Agile development methods have the aim to support the adherence to deadlines through frequent releases [1].
Furthermore, it was attempted to work out the differences within companies between the perception of development and management of agile development (AD) methods. Therefore, always two people of a company - one from management and one from the development team - was tried to be contacted for an interview. It was shown that the differences were not as high as expected.

4.2 Survey Setup and Procedure

A total set of 400 software development company addresses were collected. Unlike in [93, 96] the contacts were retrieved via the author's private and job-related network. A sample of 100 companies were contacted where 42 companies agreed (42% acceptance rate) in participating in the survey. The initial plan was to reach two people of each company, a manager and a developer. This was not possible with 23 companies (54.8%) due to lack of time, vacation of staff members, or simply because the interview partner occupied a role as a manager and developer. 19 companies (45.2%) agreed in the double interview which lead to a total number of 61 interviews (see overview Table 4.1). There were slightly more developers than managers. 25 (41%) developers, 14 (23%) managers and 22 (36%) hybrids - developers who also occupied a manager role - were reached. This distribution led to an overall "developer to manager ratio" of 1.3 where the hybrids counted for both. The survey started on 29th of July and ended on 25th of August 2008 and was conducted via telephone. The questionnaire which was used for the survey consisted of four sections:

- **General Information:** Information about the size of the company and the average size of the developer teams.
- **Agile methods and Practices:** Existence of knowledge or at least awareness of agile methods and practices, application, experiences, and problems of agile software

[1]test-first approach is considered cheap to implement since no extra tools are needed and only the developer himself is involved and 'just' has to work on his coding habits, in contrast to pair-programming where two developer and the approving management is involved

total set	400	random sample	100
comp. reached	42	success rate	42%

total interv.	61	single interv.	23
double interv.	19	manager interv.	14
developer interv.	25	hybrid interv.	22

Table 4.1: Overview of the survey participation.

development methods in general and pair-programming in particular. Furthermore, it was determined which of the easiest and inexpensive to implement agile development practices [58, 20] such as the test-first approach or continuous integration are used.

- **Future plans:** In this section the future plans to introduce agile methods and practices, especially pair-programming, are determined, and the reasons why agile methods are neglected if there were no plans to adopt to them.

- **Perception:** Finally, the personal perception to adherence to delivery dates in the software industry was collected in a passive and active role (as client and as a contractor) as well as the perceived overall tendency in the software industry. This can be seen as an indicator for a needed change in the software development process.

4.3 Company Characteristics

According the European Commission [97, 98], companies are categorized into micro, small and medium enterprises (see Table 4.2). Every company which is beyond the definition of medium size is considered a large enterprise. The distribution of the participating companies is 19% micro, 28.6% small, 11.9% medium and 40.5% large enterprises. In Table 4.3 one can see the distribution of the survey participating companies according their size. Interesting that in the sample more than 40% are large companies but only 5% medium ones. The company categorization, their maximum, minimum, and average development department size as well as the percentage below and above the average can be seen in Table 4.4. This shows that a 66% of the large categorized companies have a development department below the average - the size of the development department does not scale linear with the company size. In Table 4.5 the average project team size can be seen for each company category. It is also clear that the team size does not scale with the company size but it is remarkable that there is a strong tendency to smaller teams as 92.6% of the large companies have teams below the average size of 13. Asked for the average team-size the participants named 11 different categories. The team-size category, the average team-size, the absolute number of mention, and the percentage can be seen in Table 4.6. It shows that 78.7% have an average team-size between 2 and 6 developers. 11.5% have an average team-size above 7 developers and 9.8% of the companies are one-man companies or only have one developer. Statistically 90.2% of the companies have

Category	Headcount	Turnover
micro	< 10	$\leq$ € 2 million
small	< 50	$\leq$ € 10 million
medium	< 250	$\leq$ € 50 million

Table 4.2: Company categorization according the EU commission [97, 98].

Category	Absolute	Percentage
micro	8	19.05%
small	12	28.57%
medium	5	11.9%
large	17	40.48%

Table 4.3: Company categorization of survey participants.

Company	Development Department Size				
Category	min.	max	∅	below ∅	above ∅
micro	1	4	3.1	50%	50%
small	6	30	14	47.1%	52.9%
medium	1	25	12.1	57.1%	42.9%
large	3	200	54	66.7%	33.3%

Table 4.4: Company size vs. development department size.

the ability of applying pair-programming. Even micro companies are eligible to practice pair-programming since 60% are above the average team size of 1.75 (see Table 4.5). The five different fields of development activities the companies practice can be seen in Table 4.7. The majority of the companies (88.1%) have their main focus on commercial development. This means that their developed software is ordered and to be used by customer. 7.1% of the companies put themselves in the ”research and prototyping” category which means that they do research in a certain field and develop prototypes and prove of concept software for the customer. 4.8% of the companies develop for in-house use which means that the development department is concerned with projects only for the use in the own company, and finally, 2.4% are concerned with developing prototypes for in-house solutions which are outsourced for completion.

Company	Average Team Size				
Category	min.	max	∅	below ∅	above ∅
micro	1	2.5	1.75	40%	60%
small	1.5	5.5	3.5	76.5%	23.5%
medium	1	10.5	5.75	85.7%	14.3%
large	1	25.5	13.25	92.6%	7.4%

Table 4.5: Company vs. average team size.

Team size Category	∅ team size	absolute mentioned	% mentioned
1	1	6	9.8%
1-3	2	9	14.8%
1-5	3	20	32.8
2-6	4	7	11.5%
3-6	5	3	4.9%
1-10	6	9	14.8%
6-7	7	2	3.3%
1-15	8	2	3.3%
1-20	11	1	1.6%
1-30	16	1	1.6%
1-50	26	1	1.6%

Table 4.6: Average team-size, named size-categories and number of mention (absolute and percentage).

Development category	absolute	%
commercial	30	71.4%
research and commercial	6	14.3%
research and prototyping	3	7.1%
in-house and commercial	2	4.8%
in-house and prototyping	1	2.4%

Table 4.7: Fields of development activities of the participating companies (absolute and percentage).

4.4 Agile Methods and Practices

A majority of the interview partners (77%, or 47 people) claimed to have a general knowledge about agile software development methods. The rests (23%, 14 people) claimed to know nothing about this topic. Of those aware of agile methods, 68% (32 people) said that they would personally support agile methods or at least agile practices. There were 32% (15 people) non supporters of agile methods. Of those who did not know agile methods, 14.3% (2 people) claimed that they are personally supportive after a short introduction into agile software development methods during the interview. The desire for more information about agile methods was expressed only by 29.5% (18 people). 70.5% (43 people) stated that there is enough information available if one is willing to read up on this topic. The hit-parade of known agile methodologies can be seen in Table 4.8 where those people aware of agile methods were asked to name methods known by them. Extreme Programming (XP) was mentioned by 46% and Scrum by 32.8% which reflects the popularity of these methodologies. No other method mentioned by the survey participants exhibits such a high percentage of publicity. Only Test Driven Development (TDD) and Rapid Application Development (RAD) had a slight higher rate than the rest. Feature Driven Development (FDD), Open Unified Process (OUP), Microsoft Solution Framework (MSF) and Model Driven Architecture (MDA) were only mentioned by one person on average. Furthermore, the data shows that 23% of the participants made no distinction between agile practices, e.g., pair-programming, refactoring, and agile methodologies like Scrum and XP. The traditional Waterfall-Model, the V-Model as well as the Spiral-Model were claimed to be agile methodologies by only 5% of the participants. This shows that there is a general knowledge about software development methodologies. One goal was to find out whether there is a significant difference between managers and developers. This difference did not prove to be as high as expected. The data show that only with the methods XP and Scrum there was a difference, where XP was known by 53.2% of the developers and by 38.9% of the managers. Scrum was mentioned by 29.8% of the developers and 33.3% of the managers. For the rest of the methodologies developers and manager do not differ significantly. This also applies to the confusion between agile practices which were named as methodologies as well as the non-agile methods named as agile ones. It turned out during the survey that 54% made no distinction between the terms “iterative” and “incremental”. This indicates that those terms should be set right in context with development methods.

4.4.1 Reasons why Agile Methods are Not Adopted in Practice

When the participants answered “No” to the question whether they believe that agile methodologies are applied in practice, they were asked about the reasons for their belief. The supposed reasons can be seen in Table 4.9. “Lack of knowledge” is on top with 27.9% followed by a “negative attitude of the management towards agile methods” with 26.2% and “lack of time” with 16.4%. On the fourth place the already “established process” with 14.8% was mentioned followed by “lack of experience with agile methods” with

Method names	total	Dev.	Mngr.
XP	46%	53.2%	38.9%
Scrum	32.8%	29.8%	33.3%
TDD	4.9%	4.3%	5.6%
RAD	3.3%	4.3%	0%
FDD, Crystal, OUP	1.6%	2.1%	2.8%
MSF	1.6%	2.1%	0%
MDA	1.6%	0%	2.8%
practices named	23%	23.4%	27.8%
non-agile named	5%	4.3%	5.6%

Table 4.8: Named known agile methodologies.

Reason	total	Dev.	Mngr.
lack of knowledge	27.9%	29.8%	22.2%
management refusal	26.2%	23.4%	27.8%
lack of time	16.4%	12.8%	16.7%
existing process	14.8%	8.5%	16.7%
lack of experience	8.2%	8.5%	5.6%
customer refusal	6.6%	4.3%	8.3%
risk too high	6.6%	6.4%	5.6%
too costly	4.9%	4.3%	5.6%
company/team size	4.9%	6.4%	5.6%
resistance/fear	4.9%	6.4%	0%
tradition	3.3%	4.3%	2.8%
change takes time	3.3%	4.3%	5.6%
no/bad risk management	3.3%	0%	5.6%

Table 4.9: Supposed reasons why Agile Methods are not adopted in practice.

8.2%. “Lack of known benefits”, “bad reputation”, “customer availability” and “no need” were equally mentioned by one 1.6% (1 person). Interesting is, that again the developers' and the managers' opinion did not differ significantly from second place on. Both think that the upper management refuses agile methodologies. The reason could be that the managers the interviews were conducted with were close to the developers in the hierarchy (have been developers once) or still act as developers in their current position.

4.4.2 Adopted Agile Methods and Practices

The question whether agile methods were adopted or tried to be adopted in their company or team was answered positively by 44.3% of the total participants, 42.6% of developers and 52.8% of managers (see Table 4.10). Interesting is, the “unconscious adoption” of agile methods claimed by some participants. Of those who said they had adopted an agile methodology but unconsciously were only 1.6%. Of those who said they had not adopted agile methods claimed that it must have been done unconsciously by 6.6%. 6.6% of the adopters said that they only adopted certain practices but no distinct method. Nevertheless they said they had adopted an agile method. This shows that there is little distinction between methodologies and certain practices. 4.9% of the non-adopters said that they would have implemented certain agile practices. 55.6% (15 people) of the adopters said that the implementation went generally well (21.3% of developers and 33.3% managers). Asked for details what did work well, 60% of them (9 people) gave an answer which was not very detailed but lead to 13 positive points mentioned (see Table 4.11). The participants were also asked to name experienced issues during the adoption of agile methodologies. Fifteen adoption issues have been identified and are shown in Table 4.12. The most severe issues named by people were (a) the omission of unit-tests due to time pressure, (b) the general increase of time pressure as well as (c) the low discipline to adhere to the agile methodology by the team members. These were followed by the “resistance of developers against the agile method”, the “unwillingness of the customer to pay for the extra effort of unit-testing”, as well as the “daily stand-up meeting”, all with a percentage of 7.4%. Participants were also asked which tools and practices fostering agile development are known to them (see Table 4.13). Tools to generate documentation out of source code, e.g., Doxygen or Javadoc, are used by 70.5% of the participants. Unit-tests are claimed to be used by 75.4% of the interviewees. 18% of the participants in total pointed out that they only use unit-tests when “needed”. This shows that the practice is not an obligation but merely an option. The test-first approach is said to be used by 9.8% in total, 8.5% developers and 13.9% managers. At the same time 83.3% in total, 75% of these developers and 80% of these managers admit that they rarely use it in reality. The low percentage indicates that the test-first approach is not very popular. A reason for this might be that in spite of its cheap adoption, it needs discipline and change of habits [5]. Hence this approach is easily omitted with upcoming time pressure. The continuous integration and nightly build approach are claimed to be used by 39.3% in total. The agile practice of pair-programming is known by a total of 70.5% (43) of the participants, 72.3% (34) of the developers and 63.9% (23) of the managers. Table 4.14 shows the advantages the par-

Adoption of agile methodology	total	Dev.	Mngr.
yes	44.3%	42.6%	52.8%
yes, unconsciously	1.6%	2.1%	2.8%
no, unconsciously	6.6%	6.4%	5.6%
yes, only distinct practices	6.6%	0%	11.1%
no, but distinct practices	4.9%	4.3%	5.6%

Table 4.10: Percentage of participants claiming adoption of agile methods.

ticipants (aware of pair-programming) have considered and mentioned as significant. The top three are a.) permanent review named by 65.1% , b.) knowledge transfer named by 60.5% and c.) increase of code quality named by 53.5% of the pair-programming aware participants. 45.9% of all participants and 65.1% of those who know the practice of pair-programming said that they actually use it with a distribution of 42.6% of the developers and 63.9% of the managers (an overview can be seen in Table 4.15). Additionally, all of them stated that they did not use pair-programming regularly but instead rarely or on demand. 32.1% said they only do it with complex code and 7.1% pointed out that it is only done with newcomers for tutoring purpose. No one claimed that it was practiced as written in the books [58, 88, 20]. This shows that pair-programming in spite of the accepted advantages is still not generally established in the development processes but is used on demand and mainly in debugging and with complex code where a single developer might have run into problems. On the other side in Table 4.16 the perceived reasons for not using pair-programming can be seen. Note that the numbers include those who initially said that they do not know the practice of pair-programming.

Positive points of AM adoption	nr. of people	%
more iterations	2	13.3%
rapid prototyping	2	13.3%
less documentation	2	13.3%
unit-tests (more/at all)	1	6.7%
tool support	1	6.7%
teamwork	1	6.7%
personal responsibility	1	6.7%
deadline estimation	1	6.7%
reviews	1	6.7%
knowledge transfer	1	6.7%
user stories	1	6.7%
frequent releases	1	6.7%
customer contact	1	6.7%

Table 4.11: Named topics which went well in adoption of agile methods (percentage of adopters).

Adoption issues	%
omitted unit tests (time pressure)	11.1%
increased time pressure	11.1%
low discipline	11.1%
resistance of developers	7.4%
customer unwilling to pay for tests	7.4%
daily stand-up meeting	7.4%
communication problems	3.7%
deadlocks between teams	3.7%
planning was impossible	3.7%
team-size too small	3.7%
pair-programming not practical	3.7%
short iterations impossible	3.7%
lack of agile education	3.7%
risk estimation impossible	3.7%
many things left undone	3.7%

Table 4.12: Adoption issues. Percentage of participants claiming AM adoption in practice.

Practice names	total	Dev.	Mngr.
use of doc.-tools	70.5%	70.2%	75%
unit tests	75.4%	72.3%	80.3%
test-first	9.8%	8.5%	13.9%
cont. integration/ nightly build	39.3%	38.3%	36.1%

Table 4.13: Used tools supporting agile development.

Advantage	%
permanent reviews	65.1%
knowledge transfer	60.5%
increased code quality	53.5%
speedup	16.3%
decrease of truck-factor	9.3%
increase of creative solutions	7%
better design	7%
increased motivation	4.7%
more cost effective	4.7%
less chance of interruptions	2.3%
complex tasks more easily solved	2.3%
integration is easier	2.3%
increase of output	2.3%
increased of concentration	2.3%

Table 4.14: Advantages of pair-programming (percentages of participants knowing PP).

Knowing/using	%
knowing (of total)	70.5%
knowing and using (of total)	45.9%
using (of PP aware)	65.5%
rarely/on demand usage (of PP aware)	100%
only with complex tasks (of PP aware)	32.1%
only for tutoring (of PP aware)	7.1%

Table 4.15: Percentage of participants knowing and using pair-programming and additional habits.

Reason	total	Dev.	Mngr.
no need	30.3%	10.6%	19.4 %
too expensive	24.2%	10.6%	19.4%
no time	21.2%	14.9%	5.6%
50% productivity	15.2%	6.4%	13.9%
only with complex code	15.2%	8.5%	5.6%
lack of manpower	12.1%	8.5%	5.6%
this cannot work	9.1%	2.1%	5.6%
established process	6.1%	4.3%	5.6%
lack of experience	6.1%	4.3%	2.6%
no order/interest from management	6.1%	2.1%	2.8%
customer refusal	3%	0%	2.8%
no known benefits	3%	0%	2.8%
ego problems	3%	2.1%	2.8%
too risky	3%	2.1%	0%
no opportunity yet	3%	2.1%	0%

Table 4.16: Reasons why pair-programming is not adopted.

4.5 Future Plans for Agile Method Adoption

Asked about the planned adoption of pair-programming in the near future (meaning in the next 12 months), only 14.8% of all participants answered that they have planned to do so. 44.4% of those willing to adopt pair-programming in the near future stated that they want to do this in combination with certain agile methods (but did not name one), whereas 22.2% would like to introduce it in combination with Scrum and 11.1% with Extreme Programming. 22.2% said they would like to adopt or increasingly use pair-programming solely without other practices. A similar percentage answered to the question of adoption of agile methods in general in the near future. In Table 4.17 the overview of the percentages in total, for developers, and for managers can be seen. Asked for the reasons why not to adopt agile methods in the near future 57.4% of the participants said there was “no need”, 37.7% referred to an “established process” and 8.2% a “lack of resources” and “no time”. Similarly the reasons for not adopting pair-programming were named by 41% with “no need”, 27.9% with “established process”, 13.1% with “lack of resources” and 11.5% with “no time”. 8.2% said that they “use pair-programming to the right extent” and are satisfied but not eager to enforce or further support adoption.

Future adoption	total	Dev.	Mngr.
PP, yes	14.8%	14.9%	13.9%
PP, no	85.2%	85.1%	86.1 %
AM, yes	13.1%	14.9%	13.9%
AM, no	86.9%	85.1%	86.1%

Table 4.17: Future plans of adopting agile methods or pair-programming.

4.6 Perception of Adherence to Deadlines

In this section of the questionnaire the participants were asked to quantify their belief and expectations in adherence to deadlines vs. their experience. Both roles as a client and as a contractor should be assessed. The range was from "yes, deadline will be met" and "rather yes, deadline will be met" over "rather no, deadline might be broken" to "no, deadline will be broken" (see Table 4.18). The average experienced perception in the role as a client in adherence to keeping deadlines was 2.7 and as a contractor was 2.1. The ratio of perceived anticipation vs. experience was 0.8 in the role as contractor and 0.85 in the role as a client whereas the differences between anticipation and experience were the same in both roles. One can see that the value in the role as a contractor is slightly lower which shows the tendency of having more confidence in oneself or the own company than others. Finally, the participants had to rate the provocative statement "In the software industry deadlines are generally not met" (see Table 4.19). The range was from 1 to 5 where 1 meant "fully agree" and 5 meant "fully disagree". It is interesting that 67.8% of the answers are below the mid-level which shows a tendency towards agreeing to the statement whereas only 11.9% were above. This shows that there are perceived problems in development where agile methods could be used to increase adherence to deadlines.

Adherence to	Client role		Contractor role	
deadlines	ant.	exp.	ant.	exp.
yes	18.2%	3.6%	42.4%	12.1%
rather yes	41.8%	30.9%	45.8%	63.8%
rather no	32.7%	54.5%	10.2%	22.4 %
no	7.3 %	10.9%	1.7%	1.7 %

Table 4.18: Anticipation vs. experience of adherence to deadlines (client/contractor role).

“In SW industry deadlines are generally not met” 1...“fully agree” to 5...“fully disagree”					
	1	2	3	4	5
total	10.2%	57.6%	20.3%	11.9%	0%

Table 4.19: Statement ranking (scale: 1 to 5).

4.7 Conclusion

In this empirical study Austrian companies were examined concerning awareness and adoption of agile development methods in 2008. Although there is a general awareness about agile development there is a lack of specific knowledge about agile methodologies which becomes apparent through the answers about distinct methodologies and their practical application. Statements like “we do reviews - that’s agile” or “agile development is when the waterfall process gets stuck - countermeasures are taken” show the lack of actual knowledge about methodologies although only a third of the participants spoke out the desire for more information on this topic. There is a tendency of participants to believe that an adoption of some agile practices or practices they consider as agile are enough for agile development. Furthermore, some completely ignore or are unaware about the benefits of agile methods and practices and state “although there are studies that show advantages of agile methods, I strongly doubt their meaningful application” or “this is only hype, teamwork already exists at least 50 years”. These statements support the view that they are expressed without having tested a methodology in practice. Through the application of agile development methods adherence to deadlines should be made easier, nevertheless the expectations concerning adherence to deadlines show that people tend to believe that there are general problems keeping deadlines in the software industry. This study shows that people consider as the most severe obstacles to practical application of agile methods “lack of knowledge”, “refusal of management” and “lack of time”. However, it also shows that there is no really systematic approach towards agile development in practice.

Chapter 5

Monitoring a Software Department's Transition to Scrum

This chapter discusses how the software development department of an Austrian technology consultant company changed its software development process. The department changed its software development process from a waterfall-like process to Scrum. The reasons for the change in general, were the growing demands to short-term project orderings and changes of requirements through the customer, shorter release cycles and the need for improved quality. Scrum was chosen because of the business model, the customer and the independence of development.

5.1 Initial Situation

At the time of writing the software department worked on a highly distributed and heterogeneous environment which already went through several steps of metamorphosis concerning the architecture and the used programming languages. Also the evolving and frequently changing requirements of the customer (a telecommunication company), and the fact that the system was based on a ten year old core which grew part by part and underwent massive refactoring made it a complex field for project managers, architects and developers. Prior to the change, although the software process was a waterfall-like one, the development already made use of the following modern techniques:

- Module based design
- Unit tests
- Automated tests
- Continuous Integration
- Nightly builds
- Performance monitoring

- Automated documentation tools
- Collective code ownership
- Revision control system
- Refactoring
- Coding style guidelines
- Automated code style checkers
- Frequent code reviews
- Design patterns
- Design meetings
- Open source frameworks, tools and IDEs (Integrated Development Environments)

5.1.1 Problems Before Scrum

Because of the complexity of the system there were only a few people who had an overview of the overall functionality of the system, and the interactions of its modules, and its interfaces. These few people, called architects, were not involved in the development any more. Furthermore, a few expert developers who had detailed knowledge about different core modules had emerged over the years. Those people were vital in the projects and constantly overstressed, which led to the fact that they had very little time to coach new junior developers. Since the department had to grow due to more projects, the situation did not improve. The core problems were:

- Single points of detailed knowledge
- Too much overtime for expert developers
- Problems in coaching novice coders, mostly through too much stressed experts
- Quality loss through constant project pressure
  - Increasing system complexity through lack of quality
  - Increasing costs of new features due to increased complexity
  - Quality decline of deliveries through time pressure and complexity
- Little time for vital refactoring
- Late changes of requirements occurred frequently
- Decreasing morale of the developers

As stated before, the overall software development process was waterfall-like. Changes in the waterfall process are expensive if they happen late, but in practice they occurred frequently due to change of customer requirements. This caused constant problems keeping deadlines and imposed even more overtime. The company's management decided to go agile, although it was not clear which agile methodology in the first place.

5.1.2 The Reasons & Expectations

The most important reasons to go agile were the changing requirements, the lack of knowledge transfer from the expert to the novice coders as well as the decreasing code quality and emerging bugs. Therefore a transition team was arranged which had the task to evaluate different agile methods. XP, FDD and Scrum were evaluated. The transition team consisted of the head of the development department, architects, domain owners and the head of the project management. The expectations were that through an agile software development method it would be easier to react to changes. Through small teams instead of single experts working on projects, the knowledge should have been better distributed. Also it was expected to introduce pair programming. From this practice, better coaching, an increased quality of critical solutions and constant code reviews were expected. Summarized the expectations were:

- Get rid of single points of knowledge
- Be able to better react to (late) requirements changes
- Foster knowledge transfer
- Get novice coders faster productive
- Increase quality
  - Reduce complexity through fostering system knowledge & refactoring
  - Reduce costs for new features through a simpler system
  - Improve delivery quality through better code quality
- Decrease expert coders' overtime
- Raise the developers' morale

5.1.3 Reasons for Scrum

Scrum won, not so much because of the reason against the alternatives, but a few important reasons for Scrum. The decision in favor of Scrum was made due to the working and modern development, which basically went well except the already described problems. The developers used modern tools, open source frameworks and IDEs. The head of the department as well the transition team agreed not to fundamentally change the used tools and development practices themselves. Furthermore it was a requirement to stay fully operational during the switch. The idea was to just impose an agile framework on the development to make it agile. Additionally, the success stories in the literature [4, 15, 16, 18] and the experience with Scrum in another branch of the company were points in favor of Scrum.

5.2 Transition Schedule and Scrum Roles

The transition to Scrum was decided in the time period between July 2008 and October 2008. Although the roles in Scrum were clear (Team Member, Product-Owner and Scrum-Master) it was not clear at the beginning, who would play which role in Scrum. An in-house Scrum training with certification was given to the senior staff members who were likely to become Scrum Masters or Product-Owners. It turned out that all project mangers became Product-Owners and most of the developers became team members. Some of the developers gave up their development role and became architects or Scrum-Masters. It must be pointed out that the roles in Scrum have nothing to do with positions in the company's hierarchy. In the time during November and December, in-house trainings for scrum teams were held, information about the team constellations were discussed and meetings were held to discuss questions concerning the transition to the Scrum practice. Old projects were finished in the traditional development manner. Two teams had to start their first sprints already in December 2008 due to urgent incoming projects which originally were scheduled to be already worked off in Scrum-manner. The rest of the department started with their first sprint in January 2009.

5.3 The Department's Scrum Characteristics

Since Scrum is a framework, one has the possibilities to act free within the framework boundaries. Therefore Scrum is practiced differently from company to company, or even departments. At the time of writing some special characteristics of this department were the existence of the role of Architects, Domain-Owners and the Delivery-Team who were not part of the Scrum Teams. Furthermore, the handling of software defects from the system in maintenance had to be incorporated into the department's Scrum practice.

5.3.1 Architects

The Architects were a team of three to 5 senior former developers who had an overview of the whole system and acted as a kind of in-house consultants and authorities in questions concerning the overall architecture of planned projects so that the code base would not degrade. Furthermore, they supported the project managers in creating their offers, giving technical advice, digging out pitfalls and judging the complexity of the future projects.

5.3.2 Domain-Owners

The Domain-Owners were those who were familiar with all the business processes and how the system had to be used by the customer. The Domain-Owners did not have such a deep technically insight into the system like the Architects or the developers. For the developers, the Domain-Owners were an important source of information during the project realization phase.

5.3.3 Delivery Team

With the migration of the department's development process to Scrum, a delivery team was installed so that the Scrum-teams could concentrate on the new development process. It was expected that the productivity of the development teams would slightly decrease in the first three months. The delivery team consisted of people of the former test department and had the responsibility to prepare the system integration. Before every release, it had to build the whole system, conduct the system integration and user acceptance tests and finally to prepare and fulfill the deployment at the customer's machines.

5.3.4 Additional Meetings

In addition to the 6 Scrum meetings (estimation meeting, sprint planning 1, sprint planning 2, daily scrum, sprint review and sprint retrospective), two more meetings were held in the sprints. The sprint kick-off meeting was intended to bring all Scrum-teams, Product-Owners, Scrum-Masters and the development department leaders together to start into the new sprint. Another purpose was to spread news and talk about critical events in the upcoming sprint. The kick-off usually had a time period between 30-45min and was held on the first day of the sprint. The second additional meeting was a technical meeting, where the architects, domain-owners, the development department leader and one team member of each Scrum-team met. The purpose was to talk about technical, functional and architectural issues of the project each Scrum-team was working on in the current sprint. This meeting was held to find out possible problems and dependencies between the user stories of the different projects and Scrum-teams. The length of this meeting usually was about 90min and it always took place on the third day of the sprint. This time was chosen because every team should have already finished the planning and have had the chance to go deep enough into the code to know potential functional and architectural issues.

5.3.5 Software Defect Handling

One important function of the company in question was to maintain the shipped software system. Since the software system was in operational use and was growing release by release there was a backlog of change requests, bugs due to requirements misunderstandings or missing requirements, merge/integration failures and simply wrong implemented functions. It was the maintenance department's responsibility to support the customer and cope with these defects. These defects have had to be fixed and therefore were submitted into the company's ticket system with an according priority. If defects turned out to be an implementation issue they were forwarded to the development department so that they could be planned into the sprints (sprint planning). Therefore they were called "planned defect-tickets". Usually every team got a certain amount of defect-tickets to fix in a sprint. Note that these tickets did not have necessarily something to do with the teams' recent sprints. Some of the defects were much older and were caused in the pre-Scrum era. One can see these tickets as a kind of mini user stories which have had to be finished in a sprint since they were part of the team's commitment. If the customer reported such a defect during the sprint and the priority was high enough it was immediately distributed to one of the Scrum-teams and must have been fixed prior to the current user story. These incident tickets sometimes had to be fixed within 24 hours. These tickets were called "unplanned defect-tickets" and sometimes appeared when a new release was deployed or during the going-live process.

5.4 The Performance of a Scrum Team

In this section sprint data over a period of the whole year 2009 are analyzed and commented. During this period of time the department had 6 Scrum teams, whereas all teams performed similarly, so one team's data is used as an example to describe the situation of the whole department's Scrum practice.

5.4.1 Brief Explanation of Story Burn-Down Charts

A story burn-down chart [16, 17, 18] in Scrum is used to see how the team finishes user stories during the sprint and if they are still on track with their estimated development speed. After the sprint planning, the team commits to a number of user stories with a certain amount of user story points. Each workday in the sprint, the team works on those user stories. When one user story is finished, the amount of user story points the team estimated for this story, are subtracted from the total amount of user story points to work off. So the chart should reflect the decrease of the user story points during the sprint. This decrease of the user story points is called the burn-down. The ratio of decrease depends on the size of the user story points and the development speed of the team. So the chart can be used to visualize the work speed of the team, and whether they potentially run into problems or whether they are about to reach their sprint goal too early. In both cases the Scrum Master has to act together with the Product Owner and the Team. A typical burn-down chart has the user story points on the y-axis and the sprint days on the x-axis. Just for orientation an ideal burn-down line can be drawn from the upper left - the total amount of user story points at the beginning of the sprint to the lower right where zero user story points should be left at the end of the sprint. This line can usually not be reached in reality since the granularity is the size of one user story and therefore it rather looks like steps down to zero. Figure 5.1 could be a typical story burn-down chart. One can see that

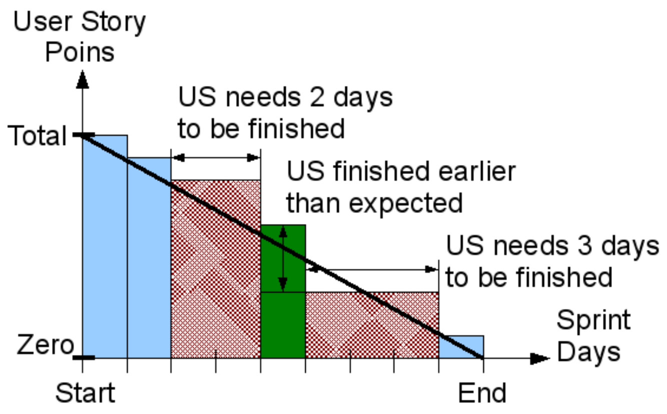


Figure 5.1: A typical story burn-down graph.

the ideal line is once under-run which shows that this user story was finished earlier than expected. Its vertical difference is bigger than with all other user stories and it brings the burn-down graph below the theoretical line. The chart also shows that the third and the one user story before the last, needed two, respectively three sprint days for completion.

5.4.2 Brief Explanation of Velocity Charts

The amount of scored user story points of a sprint is called the velocity of the team. A velocity chart [16, 17, 18] in Scrum is used to display a consolidated view of the team's scored user story points over a number of sprints. One can see the scored user stories of a sprint in relation to the other sprints. Usually with a stable team within a stable environment a team should be able to score approximately the same amount of user story points in consecutive sprints. Because of knowledge transfer and the increase of the team's performance the velocity, the team's scored user story points, should slightly increase from sprint to sprint. In reality, sick days, vacation and other events decrease the team size and hence influence the velocity, therefore such a velocity chart does not tell much about the real team performance and the sprint's circumstances but give only a hint about some irregularities during a sprint if the graph displays sudden changes. In Figure 5.2 one can see an example velocity graph. On the vertical axis, the scored user story points are plotted, on the horizontal axis the sprints. The resulting graph shows the velocity (the scored user story points of a sprint) of each sprint and irregularities can be easily recognized. For instance in the graph one can see that the velocity is increased from the first sprint to the second, then falls down in the third sprint and stays the same for the 4th sprint. In the 5th sprint and increase of the scored user story points in relation to the 4th is depicted followed by a steep decline in the 6th sprint. Unfortunately it is not possible to see whether this decline was caused because half of the team became sick or a very difficult project could not be finished in the sprint. Therefore a simple velocity chart cannot be used to judge a team's performance. To give a slightly better idea about the team's performance,

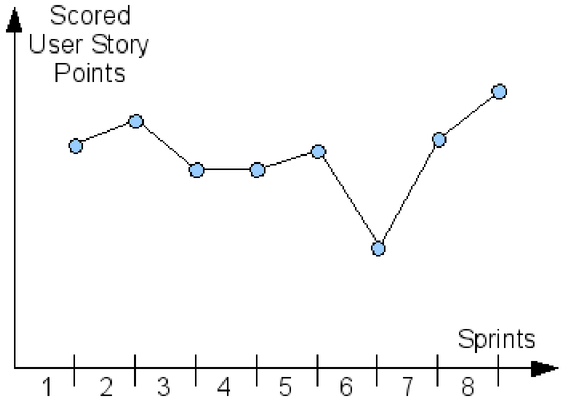


Figure 5.2: A typical velocity graph.

one can put the velocity in relation to the available person days.

5.4.3 Overview of the Sprints in 2009

The collected Sprint burn-down graphs of one year for a team can be found in Appendix E. In the discussion of remarkable sprints, it will be referred to the images in the appendix. In Figure 5.3 an overview of the 23 sprints of the year 2009 of a single team are depicted. The values on the x-axis are the sprints from the first sprint started in January 2009 until the 23rd sprint which ended in the middle of December 2009. The values on the y-axis show the user story points. All sprints longer than two weeks were normalized to two week sprints to have consistent and comparable data. The diamonds (blue) represent the committed; the vertical bars (red) represent the scored story points. The thick ramp (green) represents the trend of the velocity. Started from the third sprint trends were computed. According to Gloger [17, 18] it takes a team at least 3 sprints to accommodate to Scrum. The dotted (red) graph represents the team-size; the dashed (blue) graph represents the number of unplanned defect-fixing tasks in person days. The straight (black) line represents the trend of the unplanned defect-fixing tasks. The drawback of this graph is that it only displays the velocity which can significantly differ from sprint to sprint depending on the available person days for each sprint due to e.g., members on leave or helping other teams, etc. In Figure 5.4, the velocity is set into relation with the available person days using the formula: $Vppd = \frac{V}{PD} * 100$. The values on the x-axis are the sprints starting with the 4th sprint. The values on the y-axis represent the velocity per person days of one team. The bars (blue) represent the velocity per person days of one team from the 4th sprint in the middle of February until the 23rd sprint which ended in the middle of December 2009. The thick straight graph (orange) represents the linear trend of the velocity per person days over the depicted sprints. The velocity in context with the available person days provides a more realistic view on the team's achievements per sprint. It is noticeable that the trend in both graphs has ascending slopes. The velocity trend has an

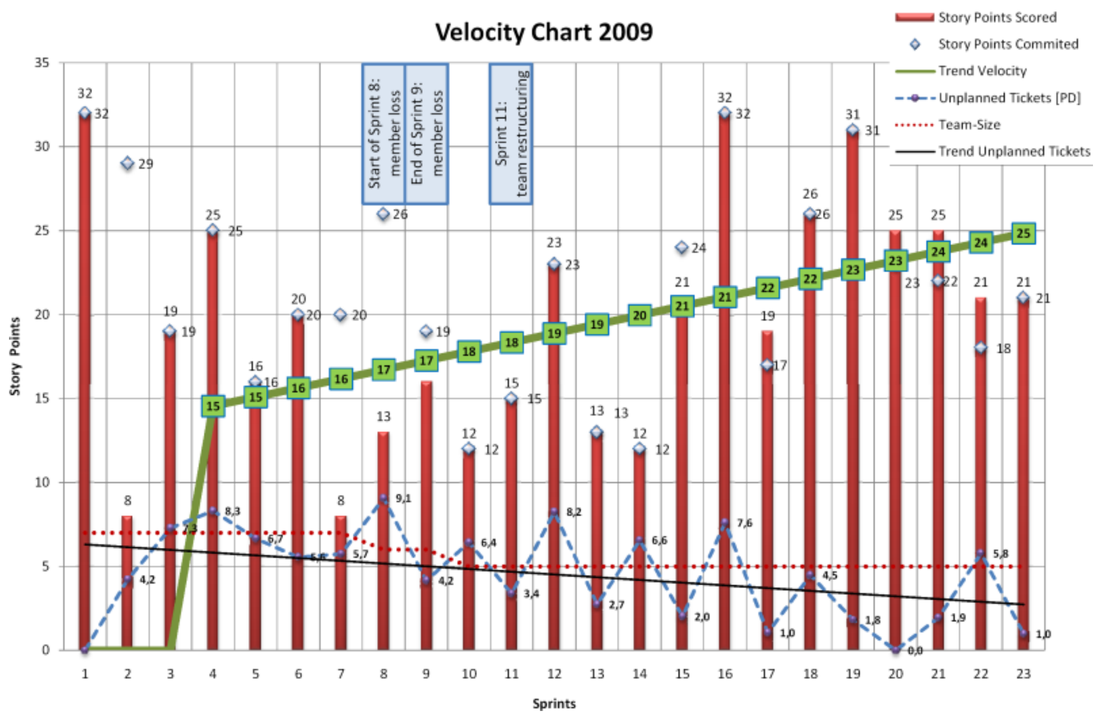


Figure 5.3: One team's velocity graph over the 23 sprints of the year 2009.

increase by 67% and the trend of the velocity per person days has an increase of 70%. It is very interesting to see that in Figure 5.3, the first sprint was fabulous followed by a second sprint where less than a third of the committed user story points could be scored which indicates a problem in the correlation between the team's reference user story and the user stories which were to be done in this sprint. It is normal that there is a break-in of the velocity in the first sprints. In the following sprints the team reacted to this and sprint two and three in the velocity graph show the leveling process and the recovery of sprint two. Furthermore one can see that in the first half of the year, the commitments were not met four times, in the second half only once. The three consecutive times in the first half (sprint 7, 8, 9), where the commitments were not met correlate with team restructuring and a high rate of unplanned defect-fixing tasks which had to be done additionally to the committed user stories. In the second half, in 4 sprints more was achieved than committed. However, the most interesting sprints are number 12, 15, 16 and 19 which will be discussed in detail in the following section. It is also remarkable, that the last three sprints of the year, although better than the average of the first half of the year are less productive than the sprints around sprint 19. This can be set in context with the upcoming end of the year but also again to the increased unplanned defect-fixing tasks in sprint 22. It is also visible that the trend of the unplanned defect-tickets decreased over the year which is an indicator for an increased development quality.

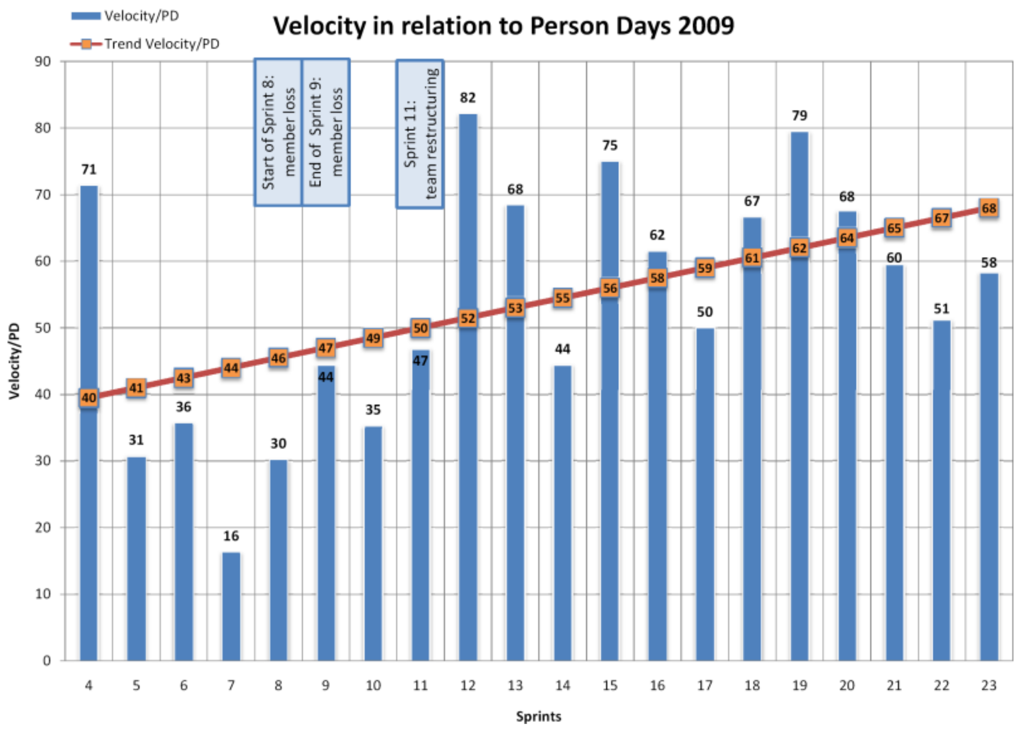


Figure 5.4: The velocity in relation to person days.

5.4.4 Discussion of Selected Sprints

Of the 23 sprints of the year 2009, 7 were selected for interestingness and grouped by characteristics of velocity and interesting issues that happened during these sprints.

Low velocity & sprint goals missed

The sprints 7, 8 and 9 were the three consecutive sprints in the first half of the year where the sprint goals were missed. This had several reasons. In sprint 7, the lowest velocity was scored, where only 40% of the committed user story points could be scored. First, the Product-Owner reported changes for the user stories for the running sprint and also had some change-requests for user stories from the previous sprint. The second reason was that defects of already reviewed user stories of the previous sprint popped up. These user stories had to be fixed. On top of this, additional defect-fixing tasks which had nothing to do with this project with an amount of 5.7 person days had to be done. At that time, unplanned defect-fixing tickets were not considered although the experience of the previous sprints showed a high burden of unplanned tasks. Third, the planned leave of some team members in the first week of the sprint lowered the theoretical development power, which can be seen in Figure E.7. This had not been considered in the team's estimations and commitment for this sprint. Finally, team conflicts showed up concerning the development power of one team member despite that team member being coached by two other

team members.
Sprint 8, although it was one day shorter (9 days) than a normal sprint, was slightly better than sprint 7. Interestingly, the team was eager to make up for the suboptimal performance of sprint 7 and committed to 30% more user story points than in the previous sprint. In the end they only scored 50% of the committed user story points (Figure E.8). The main reason was that through requirements changes reported by the Product-Owner, a couple of user stories were rendered obsolete after the sprint review. It was the Scrum-Master's mistake not to stand against this practice. Of course this would not have changed the outcome, only the statistics for the sprint. Furthermore, it turned out that two unfinished user stories were far more complex than estimated in the planning meeting. The Scrum-Master as well as the team failed to re-estimate these stories and communicate with the Product Owner. A further reason for the low velocity of the team was that a team member left the company and did not finish the committed tasks. The year's highest burden of unplanned defect-fixing tasks was assigned to the team, with an amount of over 9 person days which were done additionally. An impediment with the development machine cost another 1.9 person days of development power. A sprint external Product-Owner directly accessed team members without the knowledge of the current Product-Owner and the Scrum-Master. This behavior drained development power of about 1.5 person days. This only could happen, because the Scrum-Master was not collocated with the team and was informed by the team members only at the next daily Scrum. This communication deficiency was an issue for a couple of sprints but could be settled within the first half of the year. Summarizing sprint 8 was not that bad, since a lot of work was achieved by the team in spite of all the problems.
Sprint 9 (Figure E.9) was better than the sprint 7 and 8, but again, the sprint goal was missed by one user story (three user story points). Due to the order from the management to reduce the collected overtime of the team members, the sprint length was reduced from the initially planned 15 days to a 9 day sprint (one day shorter than the normal sprint). The reasons for not meeting the sprint goal were similar to the reasons of sprint 8, but did not have that negative effect than to the previous sprint. 84% of the committed user story points could be scored by the team. The problems were that the development power was decreased through members on leave by 5 person days, defect-fixing tasks turned out not to be reproducible which wasted development time and three user stories hat to be descoped due to unclear requirements where already development time was spent for them. The unplanned defect-fixing tasks with an amount of 4.2 person days were the half of the previous sprint's burden, and were done additionally. Again, there were troubles with another sprint external Product-Owner who drained development time from team members without telling the current Product-Owner and the Scrum-Master. The interesting part of this was that the Product-Owner who complained about this practice in the first place, did the same in another sprint. This indicates a lack of farsightedness, team work and communication among the Product-Owners. During the sprint, the requirement changes reported by the Product-Owner, forced the team to work until the last sprint day. At the sprint's end another team member left the company and reduced the team size to 5 team members, which had a negative impact on the team's velocity on contrary to the

first leave, although the member's main skills were not in development.

High velocity & more problems

Sprint 12 (Figure E.12) was the sprint with the year's highest velocity per person days (see Figure 5.4). The team which was restructured in sprint 11 (2 members were exchanged for the sake of knowledge transfer) worked euphorically and scored 23 story points. The development power was constantly decreased due to the vacation of one team member and another was helping out another team half of the sprint. The high burden of unplanned defect-fixing tasks with an amount of 8.2 person days and a user story which was added during the sprint did not break the team's commitment although it had to work until the last sprint day to finish all user stories.
Sprint 15 (Figure E.15), although the sprint goal was missed by three user story points, also had a very high velocity per person day rate (see Figure 5.4). The reason why story points were missed, were that the Product-Owner had planned and sold a project which had not been properly estimated by architects. The team estimated much more user story points in its first encounter with the project. Furthermore there was a communication problem between Product-Owners because the team was initially exclusively planned for one project during sprint 15 but due to a deadline had to finish a user story which did not fit in to the previous sprint. The team was hustled by the Product-Owner to commit to the last user story which was known to be quite challenging, since it was a big one. The Scrum-Master asked the team and the Product-Owner to split the story but both parties agreed that this was impossible. The team's development power was decreased through the vacation of one team member, but at least the unplanned defect-fixing tasks kept in limit with an amount of 2 person days. Towards the end of the sprint the team and the Product-Owner realized that the user story could not be finished and therefore was split but not descoped by the Product-Owner and got negative credits by the department leader. In the sprint retrospective the team stated that it was rightly punished for its own naivety, not to split the story in the planning meeting and to agree to commit to the whole final user story.
Sprint 16 (Figure E.16) was "the horror sprint" of the year 2009. At first glance, it looks impressive in the velocity graph. One can see an increase of 53% (11 story points). If this is compared to the value with the velocity per person days one notices a decrease of 17% (13 story points per person days) which indicates, that there went something wrong. Almost everything went wrong in sprint 16, except that the team met the sprint goal and achieved the highest velocity aside the first sprint. In the estimation meeting, which took place one day before sprint start, the team communicated that it was impossible to implement all needed functionality for the user stories the Product-Owner planned for this sprint. Obviously no architect had checked the project for feasibility before the project was sold and scheduled. The team's statement to the user stories was not accepted by the Product-Owner, since a going-live was scheduled for the consecutive sprint. The Product-Owner's opinion was that the team's commitment did not play a role and the project must be finished anyway. The team and the Scrum-Master did not accept this and went with the

Product-Owner to the department management. Unfortunately the department leader was on vacation and his substitute meant that all other teams were fully occupied and could not take over some of the user stories. Therefore he was not able to do anything about that. The promises of the management to descope functionality and find a solution for such a critical situation were not kept. The team was additionally assigned one unplanned defect with unknown origin and potential high but unknown priority which took 7.6 person days. This had a sustainable negative effect on the team's and the Scrum-Master's trust into the management of this department. The team agreed to do its best but did not commit to the amount of story points demanded by the Product-Owner. Furthermore the team revealed serious architectural shortcomings of the user stories, but nonetheless came up with a simple but unesthetic solution which should be scheduled for refactoring in an upcoming sprint. But the architects of the department did not agree with this solution, so the technical meetings for the affected user stories drained valuable development time. Until a manageable solution for these problems were found it lasted until the middle of the second sprint week. Of course all independent features of the sprint's user stories were implemented until that time, otherwise it would have been impossible for the team to finish. In the second sprint week, the Product-Owner went on a long planned vacation. A developer from another team supported the team from Wednesday of the second sprint week until the sprint's end. This can be seen in the sprint's burn-down graph, where the development power went over 100% (see Figure E.16). Additionally, the team agreed to come in at the weekend to be able to finish the user stories, and already planned vacations of two team members were deferred to a later time. All in all, there were many mistakes made. First the Scrum-Master did not escalate the problem to the next higher management hierarchy when help was refused by the department leader. Second, an additional ticket was assigned to an already "drowning" team by the department leader's substitute. Third, architectural solutions were unnecessarily complicated and a solution was delayed, since it could not be agreed to an unesthetic but simple solution suggested by the team. Fourth, the additional developer was assigned too late. Fifth, the Product-Owner refused to think about descoping functionality and went on vacation whereas the team deferred their vacation dates. Only the Product-Owner's substitute agreed at the end of the second sprint week, to a temporary partial abandonment of functionality until the next release. Nevertheless the team finished the necessary amount of user stories which look nice in the statistics but had a very high price in reality. Unfortunately, it was not possible for the purpose of this thesis to get access to the actual booked hours spent on this project which would relativize this Pyrrhic victory.
Sprint 19 (Figure E.19) had the highest velocity and the highest velocity per person days in this year. The unplanned defect-fixing tasks with an amount of 1.8 person days were low compared with the rest of the year. The development power was decreased through one team member's leave during the whole sprint. The only drawback was the high dependencies between the user stories, so their finalization moved to the sprint's end, which can be seen in the burn-down graph. Comparing the burn-down graphs from sprint 19 until sprint 23 (refer to Figures E.19, E.20, E.21, E.22, E.23) one can see that the actual graph (blue) moves closer to the planned (green) graph towards the ideal line in the

middle, which indicated that the user stories could be worked off as planned in the sprint planning, with no interruptions through unplanned tasks or other impediments.

5.4.5 Achieved Improvements

The Scrum Masters identified certain achieved improvements on different aspects of the software development process through the department's transition to Scrum, which are briefly described here.

Team work Through Scrum, real teams were formed from a former loose group of developers who were put together. This could be seen at the change of the daily Scrum Meeting. At the beginning people tried to work in their field of interest and expertise. Therefore, if their stuff was done at one user story, they wanted to move on to the next. From the 5th sprint on, people started to work together on a user story and it happened that two people or more solved one task. Furthermore they realized that helping other team members and working as a team increased the quality.

Knowledge transfer This point is tightly coupled with the previous one. Through real team work and the awareness for the team, the knowledge transfer increased greatly. This was not possible before Scrum since the experts were always overburdened and did not have time coach new members in the department. Since Scrum it was possible to integrate newbie within about 3 or 4 weeks and getting them productive. It turned out in retrospectives that it was perceived positively when a team expert was on vacation, which gave the team the possibility to show what they are able to achieve. When this happens, knowledge transfer can be considered as working and the department is prepared to grow another team.

Code quality Again, intensive team work made it possible to increase the code quality through frequent code reviews in the teams. Peer reviews automatically happen in pair programming and coaching sessions and became standard in the teams. For every user story, one team member was determined by the team to be responsible to organize a short code review. This was also recorded in the department's definition of done.

Overtime Before Scrum, the expert coders were constantly overburdened with projects in the fields of their expertise. Through the fact that the projects were worked off by whole teams and due to the knowledge transfer in the teams, it was possible to ease the overtime peak for the experts and reach a uniform level of utilization for all team members.

Vacation planning Through overlapping skill sets in the teams, it was easier for the team members to plan and go on vacation and this also in a shorter term. Furthermore it proved to be convenient that the team in agreement with the Scrum-Master were empowered to authorize vacation applications within the limit of 14 days. Of course it must have been considered that the team was able to fulfill its work and if critical projects have already been scheduled to a full team those vacations were not granted.

5.4.6 Suggestions for Further Improvement

Estimation meetings A big problem which needs to be solved is that most of the times the estimation meeting takes place one day before the first sprint planning meeting. This is too short to think about the introduced user stories and peak into the code base to identify potential pitfalls and the architecture. During the 23 sprints of 2009 it happened twice that an estimation meeting was held in the last week of the previous sprint. The Product-Owners' excuse is that there is no time since the customer orders the projects in such a short-time before the sprint, that it is impossible to conduct an estimation meeting earlier than one day before the sprint start.

Lack of available information It happened two times that a user story had to be canceled due to lack of information. In 80% of the sprints not all requirements for the user stories were available nor was it possible to get in touch with the customer to get access to a person who was able to define the requirements. This is the reason for many delays of the user stories' finalization until the very end of the sprint, which poses a high risk that bugs and shortcomings in the implementation which are found in the sprint review are almost impossible to fix in the same sprint and are deferred to the consecutive sprint.

Requirement changes during the sprint This point is tightly coupled with the previous one. Due to requirements changes during the sprint, user stories are delayed until the end and valuable development hours are wasted. It happened more than once that a user story was implemented twice or three times until the requirements were fixed. The problem is, that most of the time these requirements changes were necessary otherwise the user stories were not shippable. Especially with short term and high priority projects the requirements have to be very well defined. Having a high priority project and no idea what the requirements are is a clear contradiction.

Priority problems In about 50% of the sprints in the year 2009, more than one project was done in a sprint. Longer projects lasted for one and a half sprint. Other projects were so small that a team was not fully utilized in a sprint. Often priority struggles between different Product-Owners occurred during a sprint about whose user story had higher priority. Furthermore, it happened that a Product-Owner who had a project with the team in previous sprints, wanted to have some bugs fixed or changes for already reviewed user stories. He directly accessed the team members without the knowledge of the other Product-Owner or the Scrum-Master. The reason for that was typically a lack of communication between the Product-Owners.

Unplanned defect-fixing tasks These are activities for fixing software defects that occurred in the software system which was in operation at the customer's site. These defects are reported by the maintenance department to the development department, which in turn assigned them to a certain Scrum-team (see Section 5.3.5). The problem is that these activities are not trivial and sometimes very time consuming. In one sprint (see the dotted (red) graph in Figure 5.3) it took two additional person weeks. With short sprints and small teams of about 5 members, as in the department's case, this is a serious threat for the sprint goal and seriously disturbs the team's sprint. It happened more than once that the initially committed user stories were delayed to the end of the sprint (see collected burn-down graphs in Appendix E) and sometimes that the sprint goal was missed (see Figure 5.3). Since these tasks are neither estimated nor rewarded with story points, they are not visible in the burn-down charts and are ignored by the Product-Owners and the development department's managers. This has a demotivating effect on the team members, who do not get reward for doing additional work but negative criticism if the sprint goal was "missed in action". Fortunately, the trend over the year 2009 was negative and the assigned unplanned defect-fixing tasks were decreasing.

Dependencies between user stories User stories should be designed in a way that they are independent from each other. If this is not possible, they must be combined to a bigger user story and then split up into independent parts. It is very inconvenient if a user story's requirements change and this in turn requires a change in already finished user stories. This is a waste of time should be avoided in Scrum [16].

Short term high priority projects It is questionable if the kind of projects that the department realized in 2009 were well suited for Scrum. The Scrum-Masters are convinced that Scrum can be applied to all kinds of projects. However, short-term projects, which last in average not more than one or two 2-week sprints and have an immediate release and going-live date after the sprint, need a well organized and disciplined project and release management [16].

Architecture A good software architecture is the base of an expandable and maintainable system. The problem in this department is that due to the short term projects, architecture only gets a raw deal. It often happens that suboptimal solutions are implemented and never refactored. This makes the system more complex than necessary and more expensive to expand. This is a typical situation of accruing technical debts [99, 100] which have to be paid back to keep a maintainable system. The department leaders know this, but have been unable to bring up neither the budget nor the time for a serious refactoring sprint.

Scrum only in the development According to literature [101], an organization-wide Scrum transition needs between three to 5 years time. The development department has switched to Scrum successfully in 2009, but the way how projects are engaged, negotiated and managed has not been switched. Only the interaction between the former project managers and the developers has changed due to Scrum. The project manager, now called Product-Owner, has to write the product-backlog, introduce it to the team in the estimation meetings and show up at the daily Scrum. The Scrum Masters in this department are convinced that for an effective implementation of Scrum, not only the development but the whole organization must undergo the transition. Practices described in [102, 103, 104] for a successful company re-organization support this endeavor.

Development Efficiency & Effectivity Due to internal reports from the Product-Owners the efficiency of the development in 2009 was about 25%-30% less than before Scrum. This indicates a huge potential for further optimizations, but the Scrum-Masters are convinced that this is tightly coupled with the previous point, namely that Scrum is not yet established in the whole organization.

Software Release At the beginning of 2009 a separate delivery team was founded to keep the work of preparing the release away from the Scrum-teams. This delivery team was quite small, had no coding skills and had extreme workload peaks for the time of the delivery and going live dates. Furthermore it was very time consuming to coordinate the development teams to adhere to the code freeze. It frequently happened, that after the code freeze further changes had to be done since the Scrum-teams were often forced by requirements changes to finalize and review the user stories until the very end of the sprints. The review of user stories often brought up further defects in the user stories' implementations. At the end of September 2009, the delivery team was closed and the delivery process was given to the whole development department. From then on, the responsibility for the delivery process was rotated between the Scrum-teams. First evaluations showed an improvement, but still the delivery process needs too much time and resources and therefore is a hot spot for optimizations.

5.5 Scrum Evaluation

At the beginning of the year 2009 the development process was migrated from a traditional waterfall to an agile development method, namely Scrum. Between the 7th September and the 2nd October 2009, a survey with the goal to evaluate the introduction to Scrum was conducted. In the department the 36 developers of the Scrum-Teams were invited to take part at the survey. 32 of them completed the survey which is 89%. The 11% which were not taken into account consisted of 2 developers who started but did not finish until the deadline and 2 developers who did not attend due to personal reasons or vacation.

5.5.1 Goals of the Evaluation

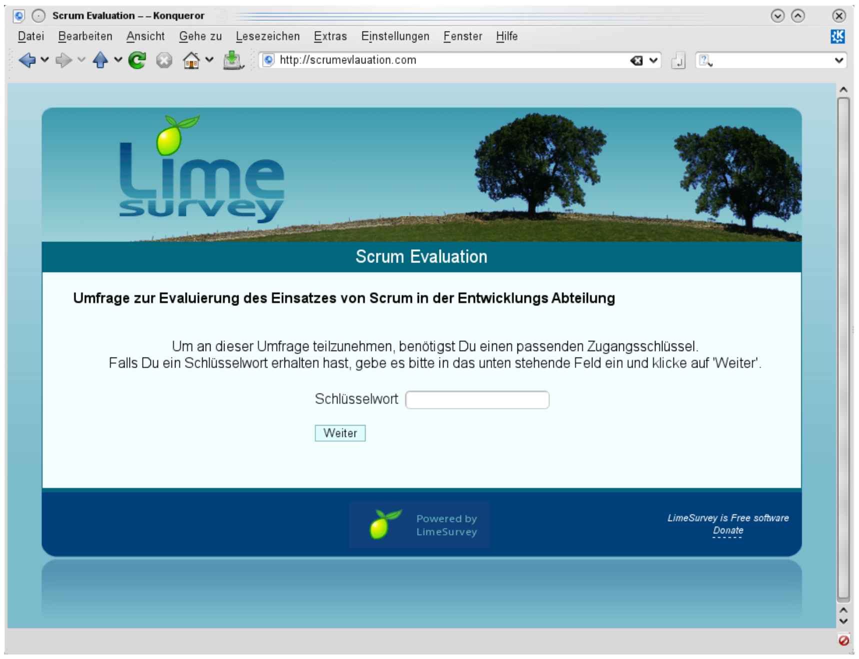
The goal of the evaluation was threefold. First it was conducted with the purpose to evaluate the software development process in the department from the Scrum-Teams' view. Second, the Scrum-Masters wanted to verify their identified fields of problems with those identified by the members of the Scrum-teams through the evaluation. Finally it was the purpose to give an opportunity to anonymously comment the overall situation and to avoid the intrinsic tendency to relativize negative statements through a face-to-face dialog. The Scrum evaluation survey was conducted in German language and the question catalog with the 121 questions is available in Appendix F. The questions, translated to English, with the associated quantifiable data can be found in Appendix G. The additional comments of the developers are omitted in the appendix, but included in the summary in Section 5.5.4.

5.5.2 Survey Design

First, one Scrum-Master wrote a basic catalog of question concerning most of the aspect of Scrum in this department. Then they met and discussed and enhanced this catalog and shaped some of the formulations. Furthermore the Scrum-Masters had the tasks to find out additional questions trough a dialog with the team members. The dialog had the purpose to find out additional problem areas of the daily work of developers which have been forgotten in the initial version of the questionnaire but it turned out, that the initial version was quite complete. Finally the catalog contained over 200 questions which was definitely too big. It was cut down to 121 questions, where a couple of these questions were conditional ones and would only be asked if a certain answer was given beforehand. The survey questions were distributed over 9 different topics concerning a.) general information, b.) the Scrum-Team, c.) the Scrum-Master, d.) the Product-Owner, e.) meetings, f.) the project handling, g.) the defect-ticket handling, h.) support of other Scrum-Teams or developers and finally i.) the delivery process. Since the Scrum-Masters only meet once a week to discuss this topic and due to vacation time, the process of finding the questions took about 4 weeks.

5.5.3 Technical Survey Setup

It was decided amongst the Scrum-Masters to conduct the survey online, instead of via printed forms. Some Scrum-Masters also were concerned with the amount of questions, the privacy and that it would be a good thing to keep the data in-house. One Scrum-Master evaluated a couple of online-tools for questionnaires and finally found a proper one, namely LimeSurvey [105]. LimeSurvey only needs a web server (e.g., Apache [106] which supports PHP5.2 [107] or higher and a database (e.g., MySQL5.x [108]. Through this it was possible to easily install the survey-server and therefore keep the data in-house. One Scrum-Master tested the created survey for consistency, executed several test runs to estimate the average duration and experimented with several methods of data analysis. An average survey run took about 10 to 15 minutes including additional comments for each topic. Two more requirements had to be met. First, anonymity and second that the survey was filled out by every participant only once, were important. Furthermore, a registration through the participant was considered to be cumbersome. These requirements were met


Scrum Evaluation – – Konqueror
Datei Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe
http://scrumevlauation.com
Lime survey
Scrum Evaluation
Schlüsselwort
Weiter
Powered by LimeSurvey
LimeSurvey is Free software
Donate

Umfrage zur Evaluierung des Einsatzes von Scrum in der Entwicklungs Abteilung

Um an dieser Umfrage teilzunehmen, benötigst Du einen passenden Zugangsschlüssel.
Falls Du ein Schlüsselwort erhalten hast, gebe es bitte in das unten stehende Feld ein und klicke auf 'Weiter'.

Figure 5.5: Scrum evaluation log-in with appropriate key.

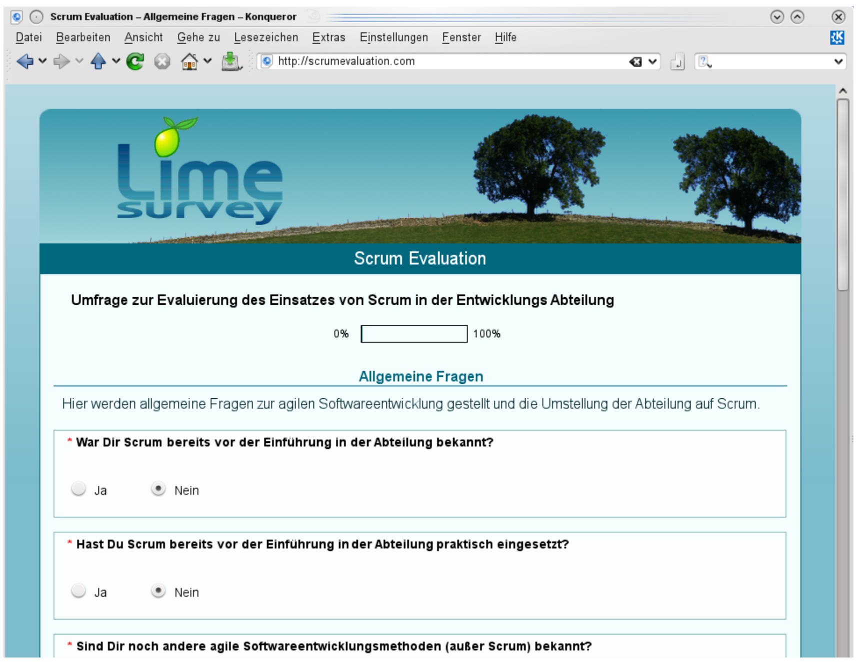
through the possibility to conduct a survey as "closed". A unique token was generated for each participant's email address and sent to the participant. It was only possible to log on to this survey with the proper token (see Figure 5.5). LimeSurvey does not store the link between the generated token and the submitted data which guaranteed enough anonymity. In Figure 5.6 the interface of the survey can be seen. At the top of the survey page, a progress bar indicated how much of the survey had already been completed. This feature was very convenient for the participants and got positive feedback. It was possible to pause and log-in with the token at a later time to complete the survey.

5.5.4 Survey Results

The complete data of the survey are given in Appendix G. In this section, the results are condensed and additional comments given by the developers during the evaluation are described.

General Information

The section "general information" was concerned with information about prior knowledge of Scrum and the attitude towards Scrum and the perceived quality of Scrum introduction in the department. 63% (20) of the developer knew Scrum before it was introduced in the department, where 6% (2) had already practical experience with Scrum. The attitude towards Scrum is rated positively by 94% (30) only 6% (2) rated their attitude as Scrum-


Scrum Evaluation – Allgemeine Fragen – Konqueror
Datei Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe
http://scrumevaluation.com
Lime survey

Scrum Evaluation

Umfrage zur Evaluierung des Einsatzes von Scrum in der Entwicklungs Abteilung

0% 100%

Allgemeine Fragen

Hier werden allgemeine Fragen zur agilen Softwareentwicklung gestellt und die Umstellung der Abteilung auf Scrum.

* War Dir Scrum bereits vor der Einführung in der Abteilung bekannt?

Ja [✓ Nein]

* Hast Du Scrum bereits vor der Einführung in der Abteilung praktisch eingesetzt?

Ja [✓ Nein]

* Sind Dir noch andere agile Softwareentwicklungsmethoden (außer Scrum) bekannt?

Figure 5.6: The first page of the Scrum evaluation survey using the LimeSurvey tool.

skeptical. The working situation had positively changed for 72% (23), remained the same for 19% (6). 9% (3) of the developers felt an aggravation of their working situation. The top three perceived advantages of Scrum are, a.) *team-work* rated by 91% (29), b.) *daily feedback* rated by 69% (22) and c.) *self organization* rated by 47% (15). Additional suggested advantages were increased knowledge transfer (4 times), shorter learning phase for newbie (once), less overtime (once), tighter co-operation with Product-Owner (once), fixed teams (once) and short notice vacation planning (once).
The top three points which need to be improved are, a.) *sprints are too short* rated by 78% (25), b.) *requirement changes during the sprint* rated by 69% (22) and c.) *impossibility of knowledge acquisition outside project boundaries*. Additional suggested disadvantages were, too many meetings/overhead (4 times), "forced commitments" due to fixed and too close release dates (2 times), Scrum is only implemented in the development but not in project engagement or management (once), team members become exchangeable (once) and projects are introduced too late which leads to stress (2 times).
94% (30) of the developers rather want to keep the Scrum process where 6% (2) would prefer to change to the pre-Scrum method again. The cases pro Scrum and against Scrum are listed in Table 5.1. 81% (26) of the developers rated that the migration to Scrum was positively accomplished. 84% (27) rated the presented information for this migration as sufficient, 16% (5) rated it insufficient. The top three positively perceived points of the migration to Scrum were a.) *timely information* rated by 53% (17), b.) *transparency of the planned migration* 41% (13) and c.) *good preparation* rated by 34% (11). The top three points which need improvement are, a.) *better training for Scrum* rated by 41% (13), b.)

earlier communication rated by 25% (8) and c.) *the migration plan could have been more transparent*.

One can conclude that the transition from the former development method to Scrum went well. The overall attitude towards Scrum is positive, although some comments from a few people are rather Scrum-skeptical. These comments, especially the negative ones, show that there is need for more information and discussions during sprint retrospectives.

Cases against Scrum	**Cases pro Scrum**
More time for knowledge acquisition	Better knowledge transfer
More efficient and goal oriented development	Shorter time for knowledge acquisition
Personal allocation was better	Self organization/ personal freedom
No discussions about the process	Prioritized user stories
Clear architectural presetting	Structured approach
Less stress	Larger projects in less time realizable
Less documentation	Less overtime
More job variation	Work is more relaxed
Feeling for success was more intense	Real team-work
Defect-ticket handling was easier	Direct influence on success
More time for testing	Testing skills are improved in the teams
More time for research	Sequential working on user stories
More intensive contact to senior developers	Transparency of project progress
Projects were longer and had better planning	Fixed teams make sense
The old approach was more agile	No roles such as developer, tester, database-guy but team member with certain skills

Table 5.1: Cases pro and against Scrum.

Scrum-Team

The section "Scrum-Team" of the questionnaire was concerned with the inner team or interpersonal issues within the Scrum-teams and how they are solved. The answers given in this section of the questionnaire reflect the atmosphere in the teams. 97% (31) of the developers are rather satisfied with the team they are working with; only 3% (1) are not. 84% developers do not, 16% (5) do have problems on a personal level in the teams and 9% (3) would appreciate a moderator to solve them. The following are exemplary comments to the question about the kind of problems which occur within the team:

- Without consulting the team, team members changed tasks they should work on until the next daily scrum.
- The attitudes, seriousness towards the commitment is individually different amongst the team members.
- Constant discussions in the team about fundamental Scrum topics are annoying.

19% (6) of the team members can imagine changing the team, whereas 81% (26) are not willing to consider a team-change themselves. 31% (10) would agree to change teams if they were asked to, but 69% (22) would strongly express their unwillingness for a team change. Concerning single points of knowledge, 81% (26) of the team members feel that they exist in their own team, 19% (6) do not see a single point of knowledge in their team. Therefore 69% (22) see an increased effort of distributing knowledge in their own team, 31% (10) see less effort of distributing knowledge in their team. 97% (31) feel that there is too little time to acquire new knowledge outside of the project scope which they feel is an innovation killer. The developers are rather satisfied with the knowledge distribution within the own team. The distribution of the work amongst the team members was rated as fair by 97% (31) of the team members. The current team size is rated as adequate by 81% (26) of the team members. Table 5.2 lists the team sizes the team members found acceptable. All team members feel that real team work happens in the teams. To optimize the work and decrease overhead 85% (27) would see an office space for each team as positive, 16% (5) think this is not necessary.

Summarizing, the developers are fond of working in teams. Although there are personal problems between team members, the majority would not like to change the team. This indicates that from a group of developers, real teams were formed. Interesting is, that in spite of increased knowledge transfer between the teams, single points of knowledge still exist after 9 months of Scrum. Furthermore it is remarkable that the majority states, that it is not possible to acquire knowledge outside project boundaries. This is a sign of too much workload.

Min. team size	**Optimum**	**Max. team size**
3-5	4-6	5-7

Table 5.2: Inquired Scrum-team size.

Scrum-Master

The section "Scrum Master" asks questions about the attitude towards the Scrum-Master, which problems occurred between the team and the Scrum-Master, and how they can be solved. 97% (31) of team members are satisfied with their Scrum-Masters. The area of the Scrum-Masters' responsibilities is clear to 81% (26) of the developers, whereas 19% (6) developers expressed their unclarity with following comments:

- The Scrum-Master's work is generally unclear.
- If there are no impediments, what does a Scrum-Master do?
- Is the Scrum-Master part of the team?
- The Scrum-Masters meet frequently with the department and development leaders - what do they talk about?
- Some work full-time in the Scrum-Master role, other only part-time and are architects or domain-owners, why?

Concerning the impediments a Scrum-Master has to solve, 88% (28) of the team members feel informed about them. 97% (31) feel that the solution of the impediments is timely achieved from the moment the impediment was noticed. The support of the team's work by the Scrum-Master is perceived as positive by 94% of the team members. 97% (31) of the team members do not have problems with their Scrum-Master. One person (i.e. 3%) do have a problem that the Scrum-Master's are too weak in keeping the team, the Product-Owners and the process Scrum-conform and would like the Scrum-Master to put more focus on the adherence to Scrum-rules. The second issue is that this person perceives the Scrum-Master as a "spy" of the management who does not bring back feedback to the team.

Although the role of Scrum-Master was rated very positively, critical comments were given concerning the Scrum-Master's work besides solving impediments. Moreover, these comments indicate that there is a potential for future conflicts. Obviously, team members perceived that the Scrum-Masters, who had mostly been developers themselves, were not on the same level in the hierarchy as developers, although the Scrum-Master role is definitely no step up in the company's career levels. Therefore the Scrum-Masters must put the focus more on transparency and information about their work.

Product-Owner

The section "Product-Owner" is concerned with the interaction between the Product-Owners (former project manager) and the development teams. Answers given in this section of the questionnaire reflect the attitude towards the Product-Owner, the problems between team and Product-Owner and possible ways to solve them. 78% (25) of the team members are satisfied with their Product-Owners. The support of the team's work by the Product-Owner is perceived by 94% (30) of the team members as positive only two people (6%) are not satisfied by the support. The area of responsibilities of the Product-Owner is clear to 94% (30) of the developers, whereas two people (6%) felt unclarity. One person expressed that he thought the main task of the Product-Owner was to present meaningful user stories and be a proxy for the customer. However, the user stories were often unclear, not consistent and without purpose. The person concluded that in his opinion the Product-Owner failed to do his job. Another person stated that the Product-Owner's responsibilities were clear only concerning the team but not known beyond that. 66% (21)

of the team members do not have problems with their Product-Owners. 34% (11) of the developers have problems with their Product-Owner, and 19% (2) of them would prefer a moderator's help to solve the problems between them. As examples of the kind of the problems between the Product-Owner and the team following points were given:

- The quality of the requirements specifications are very low.
- Some user stories are written in the estimation meeting.
- Changes of the requirements during the sprint lead to delay and additional work in the subsequent sprint.
- Prioritization of the user stories and the defect-fixing tickets, which are treated as mini user stories, are not clearly defined by Product-Owner and maintenance.
- The vision, which should be presented in the estimation meeting, often is not even mentioned.
- Product-Owners often have too little knowledge about the user stories and their interconnections which causes additional work and need time for the team to research this information.

The user stories the Product-Owner introduces before the sprint start are rated by 56% (18) of the developers as sufficiently detailed, 44% (14) think that the user stories are not formulated in enough detail. The estimation meeting and the introduction of the user stories are rated to take place too late by 69% (22) developers. Asked for the time when the estimation meeting takes place 47% (15) voted for "directly before sprint start" (e.g. the sprint starts on Tuesday, the estimation meeting takes place on the day before) and 41% (13) voted for "during the week before the sprint start". The latter correctly follows the guidelines of the department's Scrum process. 88% (28) of the developers experienced requirement changes during the sprint where 28% (9) of the developers rated these changes as "mostly", 34% (11) as "often" and 34% (11) as "rarely" with good cause. 72% (23) did not, but 28% (9) did experience a "sprint disturbing direct access to team members" through the Product-Owner. This is perceived to happen rather "rarely" or "very rarely". 87% (28) of the team member did not, 13% (4) did experience a "sprint disturbing direct access to team members" through a sprint-external Product-Owner who wants to get some additional tasks or changes done from previous sprints. But this is perceived to happen "rarely" or "very rarely". Additional comments that were given at the end of this questionnaire section are:

- The team should throw back the Product-Owner's user stories if they do not have clear requirements. Unfortunately this happens too rarely and so the team wastes time to research the requirements doing the Product-Owner's job.
- Through direct access to team members through the Product-Owner not only tasks are delayed but the whole commitment could be put on risk if this happens too often.

- The Product-Owner should earlier consult developers to get an idea what effort certain features cause that are promised to the customer.
- Between the estimation meeting and the sprint planning meetings there is not enough time to familiarize oneself with the user stories and do some research.
- Direct access to team members happens through department leaders from the maintenance.
- Additional work load for team members emerge if they are consulted for technical questions by the Product-Owners.

The data show, that here is a large potential for further optimizations. The Scrum-Masters' finding of problematic points in the department's Scrum process are supported by the data of the evaluation. The most important issues are to improve the specifications of the user stories, that the needed requirements for the user stories are brought in time and that the estimation meetings are held one week before the sprint-planning meetings otherwise it is impossible for the team members to think about the upcoming sprint.

Meetings

The Scrum framework introduces 6 meetings into a sprint and developers are known not to be fond of meetings, the section "meetings" was an important section in the questionnaire to get feedback about the meeting situation. The Scrum-meetings and their intended purpose were rated as clear by 91% (29) of the developers. 9% (3) developers mentioned some unclear points concerning the separation of the first and second sprint planning and stated that there is no increase of information between those two meetings. The realization of the meetings are rated as "rather positive" to "positive" by 84% (27) of the developers. 25% (8) (of the developers rated the number of meetings as "adequate", 75% (24) as too many. The meetings themselves are rated as "rather efficient", 22% (7) as "rather inefficient" although the time invested into the meetings are rated by 75% (24) as "rather positive", 25% (8) as "rather negative". The idea of a public demonstration of the achievements of the last sprint, additionally to the sprint review, is rated as "rather positive" by 44% (14) of the developers and "rather negative" by 56%. 75% (25) of the developers feel that there are unnecessary meetings which could be spared and gave following comments:

- If the estimation meeting takes place one week before the sprint start, the first sprint planning meeting can be omitted since the increase of information between the two meetings is marginal.
- The sprint kickoff meeting where the management and the Product-Owner tells some news and which Scrum-team is staffed for which project could be omitted.
- The sprint planning meetings, the daily scrum and the sprint retrospective happen too often.

- The estimation meeting and sprint planning on the same day is too much and does not make sense.
- There is too little time between the first sprint planning and the second.
- The whole team is rather passive during the sprint review only the one showing the user stories has something to do which is rather expensive.
- The estimation meeting and the first sprint planning meeting can be combined since they happen mostly on two adjacent days.
- It should be discussed whether all team members have take part in all meetings.

56% (18) of the team members are convinced that the goal of each meeting is met, 44% (14) are not and gave the following comments:

- In the second sprint planning often problems of technical nature are discovered or there is not enough time to go deeper.
- The requirements of the user stories are often unclear even after the sprint planning meetings.
- In the technical meetings often problems are discussed which need too much time, Therefore, no solutions can be found.
- The time between second sprint planning and the technical meeting is too long.
- Daily Scrum is only a tool to observe the team members and problems neglected.
- The time for the estimation meeting is often too short and is continued in the first sprint planning meeting.
- All influencing factors should be known by the end of the first sprint planning meeting.

Concluding, it can be stated that meetings are not popular in general and that there is a lot of potential to improve the meeting situation. The meetings must be improved concerning the efficiency and that they are held in time, which is especially true for the estimation meeting. If the requirements are not clear after the second sprint planning meeting, it should be an alarm for the Scrum-Master, the Product-Owner and the team and hence a commitment must not be given for that user story. The Scrum-meetings are intended to be work-meetings and not to sit together and talk a little bit about the upcoming sprint. It is the job of the Scrum-Master to ensure that this is understood and lived by the team members.

Project handling

This section is concerned with the general handling of projects and reflects the overall view of the current development process and where improvements can be made. 75% (24) of the developers stated to be satisfied with the project handling through the Product-Owners and the management. 81% (26) stated they were interested how projects originate and evolve from an idea to the realization through the Scrum-team, who is involved, and how the responsible people decide and how it comes to these decisions. The current project planning and realization is rated "not Scrum-conform" by 53% (17) of the developers. The period of time for the project planning and realization is rated as "rather too short" by 56% of the developers, 40% (13) rated as "adequate" and 3% (1) as "longer than needed". "Forced commitments" were already experienced by 91% (29) of the developers which means that the estimations and the commitment of the team did not play a role since a given release and going-live date directly after the sprint rendered their commitment as useless and was perceived with a frequency between "sometimes" and "quite often". A "forced commitment" means that the team has to finish all the user stories that are demanded by the Product-Owner due to a fixed deadline (going live date) immediately after the sprint. The current sprint length of 10 days was rated as "too short" by 78% (25) of the team members, 9% (3) rated as "adequate" and 15% (4) rated as "irrelevant". 91% (29) of the developers stated that during the sprint and also towards the end of the sprint they work with full energy to reach the committed sprint goal, whereas 9% (3) admit not to be that motivated and gave following comments:

- Improper information and requirement changes during the sprint as well as solutions which turn out to have flaws based on the improper information from the Product-Owner are frustrating.
- Unknown or late release- and going-live dates are perceived as demotivating.
- Different attitudes towards the commitments cause friction in the teams.
- The teams' estimations are based on inexact information. If these estimations turn out to be not exact and the commitment cannot be achieved, it is negatively judged by the Product-Owners and the management.

The teams' sprint efforts were rated as "slightly increased" by 16%(5) of the developers, as "high" by 59% (19) and as "maximum" by 25% (8). "System architectural topics are discussed with too little effort and most of the time too late" stated 59% (19) of the developers. 41% (13) are satisfied with the architectural discussion around the user stories. "Technical meetings for discussing architectural and technical issues have most of the time an influence on the already estimated user stories". This was experienced by 50% (16) of the developers. 69% (11) of those experienced an increase of complexity of the user stories most of the time. 31% (5) experienced a different influence (increase or decrease) of the complexity of the user stories. The frequency of such influences on the user stories' complexity were experienced by 41% (13) of all team members rated between “very often” by 3% (1), “often” by 6% (2) and “quite often” by 31% (10).

This section of the evaluation shows that the there is a big potential for improvements in the whole project handling chain which is rated not to be Scrum-conform by over 50% of the developers, as well as the time from the vision over the planning to the implementation as too short. The projects might be not adequate for Scrum-teams but the question is for what kind of agile methodology these projects would fit, this will be one of the future goals to find out. The projects are in general rather small, one to two, maximum three sprints long and with very little time to realization and often with fixed deadlines which lead to “forced commitments”. Here also is potential for improvement, there is no such thing as a “forced commitment” since if the team says, it cannot do it, there is no way to force the team to be convinced that it can be done and to commit itself to the full amount of the user stories. The Scrum-Master as well as the team must explicitly stand against such practices.

Defect-ticket handling

The Scrum-teams are usually confronted with two kinds of defect-tickets during the sprint, the “planned defect-tickets” which are planned aside the user stories and committed for the particular sprint and the “unplanned defect-tickets” which occur during the sprint and have to be fixed until the end of the sprint. The differences between planned and unplanned tickets are described in Section 5.3.5. This part of the questionnaire was to find out how the current practice of defect-ticket handling performs from the developer's view and where things can be improved. 81% (26) of the team members stated to be satisfied with the current way of defect-ticket handling. 56% (18) rated the way of defect-ticket handling as Scrum-conform, 44% (14) did not agree. An alternative way of handling defect-tickets would be appreciated by 38% (12) of the developers and gave following suggestions:

- Per sprint one team should be responsible for all occurring defect-tickets.
- Unplanned defect-tickets should be considered in the velocity by certain user story points.
- If more defect-tickets are expected, less user stories must be planned which is rarely the case.
- For small planned defect-tickets, no sprint planning is necessary. It would take longer to plan the solution than to fix the defect.
- In Scrum there is no such thing as ticket.

84% (27) rated the distribution of the defect-tickets among the teams as “fair”, 16% (5) did not agree. They state that the responsibilities and distribution to certain teams are not transparent and that the teams have no knowledge about other team's tickets and their interdependency. 22% (7) of the developers rated the amount of unplanned defect-tickets allocated to their team as "excessive" and a risk for the sprint commitment. 78% (25) of the team members are contend with the amount of unplanned defect-tickets. 87% (28) are satisfied with the analysis of the defect-tickets through the maintenance department. 19% (6) rated the analysis through the maintenance department between "slightly negative" to "negative" and expressed their experience by following comments:

- Defect-tickets often turn out being simple misconfiguration in the system or database flaws on maintenance's test systems. This is annoying and costs a lot of time.

- Ancient and low-priority tickets (sometimes older than a year) should simply be discarded.

- Defect-tickets especially when older, should be checked against the current version before assigned to a team, it is often the case that with the current version a bug became a feature or is already fixed.

- Often, a reproduction of the ticket's defect is not possible on the internal development and testing machines, and sometimes data for testing a special constellation is not even available.

78% (25) of the developers already experienced missing their committed sprint goal due to unplanned defect-fixing tasks with a frequency of "quite often" by 28% (9) and "sometimes" by 50% (16). 22% (7) did not experience problems with unplanned defect-tickets and their team's sprint goal. 81% (26) of the team members were satisfied by the distribution of the tickets and felt that they are assigned to the right teams. 19% (6) are not satisfied and experienced that they had to put extra effort in research to fix a defect where other teams would have had the knowledge. Furthermore they were criticized by the management or the maintenance department that the fix took that long.

The fact that unplanned defect-tickets were assigned to Scrum-teams in addition to project user stories led to many conflicts. Originally it had been expected that unplanned defect-tickets would not influence the committed user stories. This is possible to a certain extent but it happened rather often that unplanned defect-fixing tasks had an amount of over a person-week which necessarily has an impact on the committed user stories. Another demotivating factor was that if the team increased the effort and made over-hours to meet the sprint goal and additionally fix the defects, it did not reflect in the sprint statistics. Here is a big necessity to change these effects which the Scrum-Master together with the Product-Owner must pursue, that teams are not overburdened by unplanned defect-fixing tasks. The situation relaxed towards the end of the of the year which can be seen in the overall velocity chart in Figure 5.3 and related to the improved quality of code the teams produced over the year. The majority of defects to fix came from the code which was produced in the pre-Scrum era.

Inter-team support

The section "inter-team support" asked questions about collaboration between teams. With inter-team support the support of one team by a single team-external person or by a whole team is meant. This happens if technical problems arise or if too many unplanned defect-tickets pose a risk for missing the sprint goal. 97% (31) of the developers are satisfied with the mutual team support nevertheless gave some comments for improvement:

- The official way for organizing team support (request to the department leader) is tedious. Therefore, we should foster direct and uncomplicated support which is much faster.
- If support is arranged by the department leader then always discussions arise.
- Coordination of the support is a must to prevent problems with side effects working on the same module.
- A common integrated development environment (IDE) is a must, especially for the database guys.

81% (26) of the developers do not know clearly define guidelines for inter-team support and 63% (20) do not think this would make sense. 97% (31) see no problem with inter-team support and 69% (22) did not experience risking the sprint goal by helping out another team. 31% (10) of the team members have experienced a risk for missing the sprint goal through supporting another team with a frequency between "quite often" and "sometimes".

There is not much to improve with inter-team support. The teams themselves work well together and are open to help where needed. It also shows that the self organization of the teams render clear defined guidelines for inter-team support expendable.

Delivery process

The delivery team was responsible to prepare the release with all the integration testing and user acceptance tests and finally the deployment on the customer's machines. The section "delivery process" of the questionnaire reflects how well the work between the Scrum-teams and the delivery team went. 81% (26) of the developers were generally satisfied with the co-operation with the delivery team, 19% (6) were not. 78% (25) of the developers stated that the delivery team's responsibility were clear. 22% (7) expressed their unclarity through following additional comments:

- It is not clear who tests what. Should the regression test be done through the development team and if to what extent?
- The responsibilities of the delivery team are not clear since some tasks, initially on the deliver team's side are already done by the development teams like merging, user acceptance test specifications, test machine setup and partially the user acceptance tests themselves as well as the deployment.

75% (24) stated that the handover process to the delivery team was clear. 25% (8) did not share this opinion, but 100% (32) were convinced that a clear definition of the handover process would make sense. 53% (17) of the team members already experienced risking the own sprint goal due to delivery team support and this with a frequency between "often" with 3% (1), "quite often" with 25% (8) and "sometimes" with 25% (8). The idea that the acceptance test at the customer's facility should be done through the Scrum-Team, rated 40% (13) of the developers as "positive" and 60% (19) as "negative". Additional comments to these ideas were given as follows:

- If the team shall lead the user acceptance tests, a closer relation to the customer is needed.
- If the team shall lead the user acceptance tests, what purpose has the delivery team then?
- The Scrum-team has a deeper knowledge of the implemented features and therefore can better lead though the acceptance tests.
- If there is no delivery team any more, neutral and objective testing is on risk.
- If the delivery team is closed down, the regression testing will not be done any more.
- If the Scrum-teams take over the deliver team's responsibilities, this additional effort will decrease the available time for development.
- The delivery team has a better overview of the whole system and its functionality.

This section of the questionnaire reveals potential of improvement for the co-operation between the development teams and the delivery team. However, after the Scrum evaluation was finished, the delivery team was broken up and the delivery tasks were taken over by one Scrum-team which changed for each release. The members of the delivery team acted as consultants for the current release-Scrum-team and it turned out that these tasks could be handled better by the Scrum-teams than by the explicit delivery-team. The delivery team was formed to keep the responsibilities off the Scrum-teams so that they could concentrate onto development and the Scrum process, which proved to be a good decision. Nevertheless, the takeover of the delivery-tasks by the Scrum-teams should have taken place a couple of months earlier, which would have spared everyone a lot of discussions and blaming from all sides.

5.6 Conclusion

The transition to Scrum was prepared and implemented very well and a clear, believable vision for the transition was communicated. This is also reflected by the answers to the questions in the section "general information" of the Scrum evaluation survey. The first half of the year 2009 was dedicated to get used to Scrum, to form real teams out of loose groups of developers and to work on the delivery reliability. From about the fifth sprint on, team conflicts became visible and correlated with the scored user story points. What had more impact were the amount of unplanned defect-fixing tasks, the teams got assigned to. Team restructuring happened three times, at the beginning of sprint 8, and at the end of the consecutive sprint one member left the company. In the eleventh sprint, two team members were exchanged to distribute knowledge, which had a positive impact on the team. Although the size of the team went down from 7 to 5 the team became more productive until the end of the year this indicates that smaller Scrum-teams are more productive.
The second half of the year 2009 was dedicated to optimizations where also the close down of the delivery team happened. The overhead of coordinating 6 Scrum-teams to finish critical projects to a certain date (code freeze), often before the end of the sprint was too high. Furthermore the delivery team was also responsible for merging the different modules but did neither have the programming expertise nor the knowledge of the module structure to be very good with these tasks, so over the time more and more tasks were done by the teams supporting the delivery team. At the beginning of September 2009 the task of delivery and all its responsibilities were overtaken by one Scrum team, which changed every delivery. This was to decrease the coordination overhead and worked considerably better than in the old constellation.
At the beginning of this chapter, several crucial points for improvements were listed. Some of them were addressed during the year 2009, namely a.) single points of detailed knowledge, b.) too much overtime, c.) problems of coaching novice coders, d.) quality decline of delivery and finally e.) decreased team morale. Nevertheless, there are many more topics for improvement that not only correlate with the Scrum-Masters' opinion but also with the outcome of the Scrum evaluation. A transition to and implementation of Scrum is not a panacea [16] for all difficulties in software development and management, but hard work. It probably takes about three to 5 years [101] to establish Scrum in the whole organization and one must be clear about, that only parts of the positive effects using Scrum will show after a short time. Changes should be done step by step using the Deming-cycle [109]: Plan, Do, Check, Act. Only through constant review and considering feedback it makes sense to do the next step. Since there is no approach that guarantees success, the management of the organization must stand behind the decision but should be careful with statements that show too much determination to the transition, like *"Within the next year, the productivity will be increased by 25%"*, but also have the potential to demotivate and discourage the employees who primarily must implement the change. According to [104], fear, disinterest, and frustration are the main impediments for a successful change in the organization.

Chapter 6

Conclusions

For production of software in a commercial context a working software development process is vital. Agile software development methods became very popular in the last decades as an answer to the demands of short time to market and changing requirements. This thesis is concerned with the research question of how agile software development methods are applied in practice.

In the htmlButler project it was decided to use an agile development methodology. Extreme Programming (XP) was chosen, since it became very popular recently. The people centered approach of XP makes it the ideal process for small and mid-sized projects. The htmlButler project progress with its mediocre success, the team problems as well as the problems applying the XP principles made it necessary to analyze these problems and to investigate whether XP was the right choice. In Chapter 3 the goals of the htmlButler project are described and the outcome of the project as well as the problems with the introduction of XP is analyzed. For this purpose a post mortem questionnaire was conducted between the developers which showed massive problems in team and the lack of guidance through the management. The last part of the questionnaire was designed in a way that the data could be used to estimate the project success potential using the “Chaos Report” algorithm. The results of the ”Chaos Report” algorithm, together with an assessment of the manpower in research teams dealing with the same topics as the htmlButler project, clearly show that the htmlButler project was too ambitious for the given project setting (a ”high risk project” according to the Chaos Report algorithm’s categorization). If this assessment had taken place at the beginning of the project, it would have been possible to react earlier to most of the problems, and thus increase the output of the htmlButler project.
Furthermore, in Chapter 4 the question arose whether agile methods in general and XP specifically, are used and how they are applied within a commercial project context in the Austrian IT industry. This is especially interesting since it was already hard to introduce XP in a small team of computer science students, who were familiar with agile software development methods. For this purpose 100 companies of the Austrian IT industry were contacted for a telephone interview, where 40 agreed to participate in the survey. The outcome was disillusioning, in that despite the wide acceptance of agile methods in academia

and amongst enthusiastic software development adepts, the application of agile methods in commercial environments is rare. The survey showed that although there is a general awareness about agile development there is a lack of specific knowledge about agile methodologies, e.g., XP and Scrum and that there is a tendency to believe that an adoption of some agile practices or practices they consider as agile are enough for agile development. Furthermore, this study shows that people consider as the most severe obstacles to practical application of agile methods “lack of knowledge”, “refusal of management” and “lack of time”. It also shows that there is no really systematic approach towards agile development in practice. These reasons implicate that in order to foster agile software development in practice, increased efforts must be made by enthusiasts to educate not only developers but also and especially people in management positions in the software industry.
In Chapter 5, a transition of a software development department from a waterfall process to the Scrum framework was described. The reasons for the transition to an agile software development methodology and why Scrum was chosen were manifold. However, the most relevant reasons for the transition were a.) rapidly changing requirements in all projects, b.) the lack of knowledge transfer from experts to novice coders as well as, c.) the decreasing code quality and emerging bugs in the released versions of the system. After the transition, all 23 sprints of one Scrum team in the year 2009 were monitored and analyzed in the scope of this thesis. The overall velocity chart for the year 2009 showed interesting tendencies and events like a.) the transient effect of the velocity during the first sprints, b.) a correlation between team problems and scored user story points, c.) an overall positive trend of the velocity and the velocity per person days and d.) a decreasing amount of unplanned defect fixing tasks during the sprints. The main improvements that were achieved during the year 2009 are a.) established real team work, b.) improved knowledge transfer, c.) improved code quality and d.) decreased overtime for all developers. Nevertheless, there are many more topics for improvements left, the most important being a.) lack of available information, b.) requirement changes during the sprints, c.) increasing technical debt through short term high priority projects and d.) that Scrum was only established in the development but not yet in the whole organization. The conducted Scrum evaluation amongst the development teams supports the findings of the analysis. In addition, it shows that the general attitude towards Scrum and the perception of the transition to Scrum was positive 9 months after introduction.

Summarizing, this thesis contains concrete descriptions of how agile software development methods are applied in practice. To some extent, the results directly point to possible solutions for the problems that have become apparent. In order to advance the application of agile software development methods in commercial settings, the issues of team building, software project management and education of all involved parties have emerged as the most important ones. At the same time these are the most promising topics for further research.

Bibliography

[1] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. Manifesto for Agile Software Development. http://agilemanifesto.org, 2001. visited: July, 2006.

[2] Alistair Cockburn. *Agile Software Development*. Addison-Wesley Longman, Amsterdam, October 2001.

[3] Jim Highsmith, Alistair Cockburn. Agile Software Development: The Business of Innovation. *Computer*, 34(9):120–122, September 2001.

[4] Ken Schwaber, Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 1st edition, October, 15th 2001.

[5] Scott W. Ambler. Introduction to Test Driven Design (TDD). http://www.agiledata.org/essays/tdd.html, 2002. visited March 2008.

[6] Jim Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley Longman, Amsterdam, March 2002.

[7] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, Jussi Ronkainen. New Directions on Agile Methods: A Comparative Analysis. In *Proceedings of the International Conference on Software Engineering*, pages 244–254. IEEE, 2003.

[8] Barry W. Boehm. Get Ready For The Agile Methods, With Care. *Computer*, 35(1):64–69, January 2002.

[9] Mike Cohn. *Agile Estimating and Planning*. Prentice Hall, 2005.

[10] Alan S. Koch. *Agile Software Development: Evaluating the Methods for Your Organization*. Artech House, 2005.

[11] Peter Schuh. *Integrating Agile Development in the Real World*. Cengage Charles River Media, 2005.

[12] Scott W. Ambler. The Agile Unified Process (AUP). http://www.ambysoft.com/unifiedprocess/agileUP.html, June 2006. visited October 2007.

[13] John Hunt. *Agile Software Construction*. Springer, 2006.

[14] Stefan Roock. Jeff De Luca on Feature Driven Development - An Interview. http://www.it-agile.de/fddinterview_english.html, April 2007.

[15] Ken Schwaber. *Agiles Projekt Management mit Scrum*. Microsoft Press, 2007. German Edition of Agile Project Management with Scrum.

[16] Roman Pichler. *Scrum - Agiles Projektmanagement erfolgreich einsetzen*. dpunkt.verlag, 2008.

[17] Boris Gloger. *Scrum. Produkte zuverlässig und schnell entwickeln*. Hanser Verlag, 1st edition, April 2008.

[18] Boris Gloger. *Scrum - Produkte zuverlässig und schnell entwickeln*. Hanser Verlag, 2nd edition, 2009.

[19] James Bach. What software reality is really about. *Computer*, 32 Issue 12:148 – 149, 1999.

[20] Kent Beck, Cynthia Andres. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman, Amsterdam, 2nd ed. edition, December, 2nd 2004.

[21] Ian Sommerville. *Software Engineering*. Addison-Wesley, 5th edition, 1996.

[22] Andreas Spillner, Tilo Linz. *Basiswissen Softwaretest*, chapter 3, page 41. dpunkt, 2005.

[23] Barry W. Boehm. Software Engineering: R & D Trends and Defense Needs. In Peter Wegner, editor, *Research Directions in Software Technology*, number 717. Cambridge MA: MIT Press, 1979.

[24] Meir M. Lehman. Laws of Software Evolution Revisited. In Carlo Montangero, editor, *Proceedings of Software Process Technology, 5th European Workshop, EWSPT96, Nancy, France*, volume 1149 of *Lecture Notes in Computer Science*. Springer, October 9-11 1996.

[25] IEEE. *IEEE Std 1219-1998, IEEE Standard for Software Maintenance*. IEEE Computer Society, USA, http://www.ieee.org, October 1998.

[26] Alain Abran, James W. Moore, Pierre Bourque, Robert Dupuis. *Guide to the Software Engineering Body of Knowledge 2004 Version [SWEBOK] Software Engineering Body of Knowledge, A. Abran and J. W. Moore, eds., IEEE Computer Society Press, Los Alamitos, CA, 2001*. IEEE Computer Society, Los Alamitos, California, http://computer.org, 2004 edition, 2004.

[27] IEEE/EIA. *IEEE/EIA 12207.0-1996 IEEE/EIA Standard Industry Implementation of International Standard ISO/IEC 12207: 1995 (ISO/IEC 12207) Standard for Information Technology Software Life Cycle Processes*. IEEE Computer Society, USA, http://www.ieee.org, March 1998.

[28] Bennet P. Lientz, E. Burton Swansons. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

[29] ISO/IEC. *Software Engineering-Software Maintenance ISO/IEC 14764:1999*. ISO/IEC Geneva, Switzerland, 1999.

[30] Guimarares T. *Managing Application Program Maintenance Expenditures*, volume Communications of the ACM 26 (10), pages 739–746. ACM, 1983.

[31] Thomas M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, 1997.

[32] Jussi Koskinen. Software Maintenance Costs. Information Technology Research Institute, University of Jyväskylä http://www.cs.jyu.fi/∼koskinen/smcosts.htm, September 2003.

[33] Shari Lawrence Pfleeger, Joanne M. Atlee. *Software Engineering—Theory and Practice*. Prentice-Hall International, 3rd edition, August, 31st 2005.

[34] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall Advances in Computing Science & Technology Series. Prentice Hall PTR, November, 1st 1981.

[35] Marcario Polo, Mario Piattini, Francisco Ruiz. *Advances in Software Maintenance Management: Technologies and Solutions*, chapter Chapter 8 - Software Maintenance Cost Estimation, page 312. Idea Group (IGI) Publishing, 2003.

[36] Pankaj Jalote. *An Integrated Approach to Software Engineering*. Text in Computer Science. Springer, Berlin, 3rd edition, November, 30th 2005.

[37] Bijay K. Jayaswal, Peter C. Patton. *Design for Trustworthy Software: Tools, Techniques, and Methodology of Developing Robust Software*. Prentice Hall International, 2006.

[38] W. W. Royce. Managing the Development of Large Software Systems. In *IEEE WESCON*, pages 1–9, August 1970. Reprint in proceedings of the 9th International Conference on Software Engineering (ICSE-9), pages 328-338, 1987, Monterey, CA.

[39] Herbert D. Benington. *Production of Large Computer Programs*, volume IEEE Annals of the History of Computing 5. IEEE, October-December 1983.

[40] Victor R. Basili and Albert J, Turner. *Iterative Enhancement: A Practical Technique for Software Development*, volume SE-1, pages 390–396. IEEE, December 1975.

[41] Meir M. Lehman. The Programming Process. IBM Res. Rep. RC 2722, IBM Research Centre, Yorktown Heights, NY 10594, September 1969.

[42] Barry W. Boehm. Guidelines for Verifying and Validating Software Requirements and Design Specifications. In *Proc. European Conf. Applied Information Technology (IFIP 79)*, pages 711–719, September 1979.

[43] Barry W. Boehm. *A Spiral Model of Software Development and Enhancement*, volume Computer vol. 21. IEEE, 1988.

[44] Thomas Grechenig, Mario Bernhart, Roland Breiteneder, Karin Kappel. *Softwaretechnik: Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Studium, October, 21st 2009.

[45] Ivar Jacobson, Grady Booch, James Rumbaugh. *Unified Software Development Process*. Addison-Wesley Longman, Amsterdam, 1999.

[46] The Eclipse Foundation. Introduction to OpenUP. http://www.eclipse.org/epf/general/OpenUP.pdf, August 2007. visited October 2007.

[47] Scott W. Ambler. Enterprise Unified Process (EUP). http://www.enterpriseunifiedprocess.com/, September 2007. visited October 2007.

[48] Per Kroll, Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*. Addison Wesley, 2003.

[49] John Hunt. *Guide to the Unified Process Featuring UML, Java and Design Patterns*. Springer, 2003.

[50] Jim Arlow, Ila Neustadt. *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley Longman, Amsterdam, 2 edition, July, 14th 2005.

[51] Kent Beck. Embracing Change with Extreme Programming. *Computer*, 32(10):70–77, October 1999.

[52] Duane Truex, Richard Baskerville, Julie Travis. Amethodical Systems Development: The Deferred Meaning of Systems Development Methods. *Accounting, Management and Information Technology*, 10:53 – 79, 2000.

[53] Peter Naur. Understanding Turing's Universal Machine — Personal Style in Program Description. *The Computer Journal*, 36:351–372, 1993.

[54] Richard Baskerville, Julie Travis, Duane P. Truex. Systems Without Method: The Impact of New Technologies on Information Systems Development Projects. In Kenneth E. Kendall, Kalle Lyytinen, Janice I. DeGross, editor, *Proceedings of the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies in Information Systems Development, Minneapolis, Minnesota, USA*, volume A-8 of *IFIP Transactions*, pages 241–269. North-Holland, 14-17 June 1992.

[55] Watts S. Humphrey. *A Discipline for Software Engineering*. SEI Series in Software Engineering. Addison Wesley, January 1995.

[56] Watts S. Humphrey. *Introduction to the Personal Software Process*. SEI Series in Software Engineering. Addison Wesley, March, 5th 1997.

[57] Christian Schindler. Agile Software Development Methods and Practices in Austrian IT-Industry: Results of an Empirical Study. In Masoud Mohammadian, editor, *Proc. 2008 International Conference on Computational Intelligence for Modelling, Control & Automation (CIMCA 2008), Intelligent Agents, Web Technologies & Internet Commerce (IAWTIC 2008), Innovation in Software Engineering (ISE 2008)*, pages 321–326, Vienna, Austria, December 10-12 2008. IEEE Computer Society.

[58] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, US edition, October, 5th 1999.

[59] Laurie Ann Williams. *The Collaborative Software Process*. PhD thesis, University of Utah Department of Computer Science, 2000.

[60] Jim Highsmith. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing Co Inc.,U.S., January 2000.

[61] Sam Bayer, Jim Highsmith. RADical Software Development. The American Programmer Magazine, June 1994.

[62] Alistair Cockburn. *Crystal Clear A Human-Powered Methodology for Small Teams*. Addison Wesley Professional, October, 19th 2004.

[63] Stephen R. Palmer, John M. Felsing. *A Practical Guide to the Feature-Driven Development*. Coad Series. Prentice Hall International, Februar 2002.

[64] Frederick Phillips Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, anniversary edition edition, 1995.

[65] Pranjal Arya and Christian Schindler and Wolfgang Slany. Human-Agent interaction in the light of ontology sharing and large scale cooperation. In *Proc. Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2005)*, pages 158–161, Vienna, Austria, November 28-30 2005. IEEE Computer Society.

[66] Karl Neuhold and Christian Schindler and Wolfgang Slany. The htmlButler Approach: Through Shared Ontologies and Large Scale Cooperation to Enhanced Wrapper Usability. In *Proc. 6th International Conference on Knowledge Management (I-KNOW06)*, pages 35–38, Graz, Austria, September 6-8 2006.

[67] R. Baumgartner and S. Flesca and G. Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of VLDB*, 2001.

[68] M. Herzog and G. Gottlob. InfoPipes: A flexible Framework for M-Commerce Applications. In *Proceedings of TES workshop at VLDB, 2001*, 2001.

[69] Sahuguet, A. Azavant, F. Web ecology: Recycling HTML pages as XML documents using w4f. In *Proceedings WebDB, 1999*, 1999.

[70] David Huynh, Stefano Mazzocchi, and David Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In *Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland*, volume 3729 of *Lecture Notes in Computer Science*, November 6-10 2005.

[71] Wales, J. and Sanger, L. The Wikipedia Project. http://wikipedia.org. visited June 2005.

[72] Flickr. http://www.flickr.com. visited: September 2005.

[73] Goodger, B. et al. XML User Interface Language (XUL) 1.0. http://www.mozilla.org/projects/xul/xul.html, 2001. visited: January 2006.

[74] Mozilla Foundation. Mozilla.org. http://www.mozilla.org/. visited: January 2006.

[75] M. Hatala and G.Richards. Global vs. Community Metadata Standards: Empowering Users for Knowledge Exchange. In *Proceedings of the First International Semantic Web Conference.*, 2002.

[76] M. Missikoff and X.F. Wang. Consys - A Group Decision-Making Support System For Collaborative Ontology Building. In *Proceedings of Group Decision & Negotiation 2001 Conference*, 2001.

[77] M. Stonebraker and J. Hellerstein. Content Integration for E-Business. In *ACM Sigmod Conference*, 2001.

[78] W3C and Refsnes-Data. W3C RSS Tutorial, 2006. visited June 2006.

[79] Wikipedia. WikiWikiWeb. http://en.wikipedia.org/wiki/WikiWikiWeb. visited June 2005.

[80] Enrico Motta, Marta Sabou. Next Generation Semantic Web Applications. In *Proceedings of the 1st Asian Semantic Web Conference (ASWC), Beijing, China, 3-7 September, 2006*, 2006.

[81] Martin Lechner. XP Team Psychology - An Inside View. In *PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference, Lancaster University*, September, 10th - 12th 2008. PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference, Lancaster University, UK. 10th - 12th September 2008.

[82] Philippe Kruchten. *The Rational Unified Process - An Introduction*. Addison-Wesley Longman, Amsterdam, 3rd edition, December 2003.

[83] Kweku Ewusi-Mensah. *Software Development Failures: Anatomy of Abandoned Projects*. MIT Press, 2003.

[84] StandishGroup. The CHAOS Report. The Standish Group, 1994. http://standishgroup.com.

[85] Robert L. Glass. IT Failure Rates - 70% or 10 - 15%. *Software, IEEE*, 22(3):110–112, May - June 2005.

[86] Robin F. Goldsmith. REAL CHAOS, Two Wrongs May Make a Right. *IT Metrics and Productivity Journal http://www.compaid.com/*, 2008/12/17, December 2008.

[87] Norman L. Kerth. *Project Retrospectives - A Handbook for Team Reviews*. Dorset House Publishing Co Inc.,U.S., 2001.

[88] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison Wesley, 2003.

[89] Bruce W. Tuckman. Developmental Sequence in Small Groups. *Psychological Bulletin*, 63 (3)(3):384–399, June 1965. Reprint (with permission) in Group Facilitation: A Research and Applications Journal No. 3, Spring 2001 (doi).

[90] Günther Krumpak. *IT-Business in Österreich*. Bohmann, 2001.

[91] *IT-Business in Österreich 2008*. Monitor Bohmann, Vienna, 2008.

[92] IDC CEMA. Austria IT Services 2008-2012 Forecast and 2007 Vendor Shares. IDC Central Europe GmbH, August 2008.

[93] Christian Hofer. Software Development in Austria: Results of an Empirical Study among Small and Very Small Enterprises. In *Proceedings of the 28th Euromicro Conference (EUROMICRO-02)*, 2002.

[94] Kohsuke Kawaguchi et.al. Hudson - an extensible continuous integration engine. https://hudson.dev.java.net/. visited January 2008.

[95] ThoughtWorks, Inc. Cruisecontrol a framework for a continuous build process. http://cruisecontrol.sourceforge.net, 2001. visited January 2008.

[96] Wolfgang H. Janko and Edward W. N. Bernroider and Walter Ebner. Softwarestudie 2000 Eine empirische Untersuchung der österreichischen Softwarebranche. *ADV*, 2000.

[97] COMMISSION OF THE EUROPEAN COMMUNITIES. COMMISSION RECOMMENDATION of 06/05/2003 concerning the definition of micro, small and medium-sized enterprises. *Brussels, 06/05/2003 C(2003) 1422 final*, 2003.

[98] European Commission. European commission - sme definition. http://ec.europa.eu/enterprise/enterprise_policy/sme_definition, May 2003. visited: June 2008.

[99] Ward Cunningham. The WyCash portfolio management system. *OOPS Messenger*, 4(2):29–30, 1993. DBLP:journals/oopsm/Cunningham93.

[100] Kane Mar, Michael James. Technical Debt and Design Death. http://danube.com/scrum/whitepapers, 2006.

[101] Ken Schwaber. *The Enterprise and Scrum*. Microsoft Press, June, 13th 2007.

[102] John P. Kotter. *Leading Change*. Mcgraw-Hill Professional, September, 1st 1996.

[103] Jim Collins. *Good to Great: Why Some Companies Make the Leap...And Others Don't* . Harper Business, October, 16th 2001.

[104] John P. Kotter, Dan S. Cohen. *The Heart of Change: Real Life Stories of How People Change Their Organizations*. Mcgraw-Hill Professional, July, 1st 2002.

[105] Carsten Schmitz, Thibault Le Meur, David Olivier, Jason Cleeland, Amit Kumar, Evan Wills, Karolina Maneva-Jakimoska, Bob Cunningham, Jörg Schneider, Daniel Dao Quang Minh, Mac Duy Hai,Tim Wahrendorff. Lime Survey, the open source survey application... refreshingly easy and free. http://www.limesurvey.org, 2009.

[106] Apache HTTP Server Project. http://httpd.apache.org/, 2009.

[107] PHP: Hypertext Preprocessor. http://php.net/, 2009.

[108] MySQL Community Server. http://www.mysql.com/, 2009.

[109] William Edwards Deming. *Out of the Crisis*. The MIT Press, August, 11th 2000.

[110] Meir M. Lehman, Lazlo A. Belady. *Program Evolution: Process of Software Change*. London: Academic Press, 1985.

Appendices

Appendix A

Lehman's Laws

Lehman's investigation of the software process in a period of over 20 years led to his eight "laws" of software evolution [24]and are briefly described below:

- Continuing Change (1974)
  A program used in a real life environment must be adapted continuously otherwise it will progressively degrade in its usefulness since the operational context changes over time.

- Increasing Complexity (1974)
  The complexity of an evolving system increases since the structure becomes more complex when changes are made. The increase of complexity makes further changes more difficult therefore additional resources must be provided to keep the structure simple in spite of implemented changes.

- Self-Regulation (1974)
  Program evolution is a self-regulating process. Software is implemented within a wider organizational context therefore the completion of the program is constrained by the wider objectives and constraints of this context at all levels. All feedback controls the way the system evolves. The time between releases and the number of change requests due to discovered errors tend to be invariant.

- Law of Conservation of Organizational Stability (1978)
  Over the life span of a program the average rate of work tends to be constant and independent of the resources and to its development.

- Conservation of Familiarity (1978)
  Over the life span of a program the average changes in each release tend to be constant.

- Continuing Growth (1991)
  This is closely coupled with the first law. The features of a program (functional capability) must continuously increase to keep the usefulness of the program. System growth is driven by feedback from its users.

- Declining Quality (1996)
  Program quality tends to decrease. A program is built on certain assumptions and over time due to a changing environment these assumptions become invalid.

- Feedback Systems (1996 already recognized 1971)
  Evolution processes are feedback systems. Feedback plays a role in all of the laws, this was recognized in [41]. Later Studies in the 1970's showed self-stabilizing feedback system behavior. "The process leads to an organization and a process dominated by feedback," [110].

These consolidated findings formulated as "laws" show that software has an evolutionary nature and without maintenance, software would cease to be useful with the time.

Appendix B

htmlButler Project Post Mortem Questionnaire

In this Appendix the questions for the htmlButler post mortem survey are described. This survey was conducted by means of word processing files which were sent to the involved developers by email. The discussion of the consolidated data can be found in Chapter 3.

B.1 Introduction

This is an anonymous questionnaire. The purpose is to analyze the htmlButler project and find out which and how things could be done in a different way to improve the overall outcome of similar projects in future. Any information which could be traced back to you will be eliminated in the evaluation process and will neither be disclosed to public nor used in another way as previously described.

B.2 General Information

Have you been staffed full-time/part-time (e.g. 40h/week or less) to the project:
☐ full
☐ part

Did you participate the complete period (1st Feb. 2005 - 31st Jan. 2007) in the project:
☐ Yes
☐ No

Did you enjoy your time on the project:
☐ (5) excellent
☐ (4)
☐ (3)

☐ (2)
☐ (1)
☐ (0) not at all

Was your area of work in the project clear to you:
☐ (5) absolutely
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) not at all

Was the project-target clear to the developers:
☐ (5) absolutely
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) not at all

Was the project-target clear to the project- and upper managers:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) not at all

Please rate your motivation at the START of your time in the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate your motivation at the END of your time in the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate your own contribution to the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) nothing

Please rate your own ability to work independently:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) cannot work independently

Please rate your payment:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

I see myself as someone who...			
...is reserved		...is generally trusting	
...tends to be lazy		...is relaxed, handles stress well	
...has few artistic interests		...is outgoing sociable	
...tends to find fault with others		...does a thorough job	
...gets easily nervous		... has an active imagination	
...is considerate and kind to almost everyone			

Table B.1: Self assessment. Scale: 5 (“strongly agree”) ... 0 (“strongly disagree”).

B.3 Teamwork

How many developers were involved in the project:
☐ (5) too many
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) too few

How many developers participated in the project during your time:___

Please rate your capacity/ability for teamwork:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate the capacity/ability for teamwork for your coworkers:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Did you have problems with team members:
☐ Yes
☐ No

If yes, please tell the number of team members you had problems with:___
...and please describe the reason with short terms:___

Please rate the team spirit (moral/mood) of the team at the START of the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate the team spirit (moral/mood) of the team at the END of the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate the ability of the team members to work independently:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Was the distribution of work among the team members fair:
☐ Yes
☐ No

If not, did you have to work more than others:
☐ Yes
☐ No

Do you favor agile methods to traditionally software development methods:
☐ Yes
☐ No

Have you worked in a software project before:
☐ Yes
☐ No

Have you worked with agile methods before:
☐ Yes
☐ No

Have you worked in a project utilizing agile methods before:
☐ Yes
☐ No

Please rate you experience level with agile methods:
☐ (5) excellent

☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate your experience level in programming:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Did you have problems making yourself understood in the meetings:
☐ Yes
☐ No

Did pair programming work in your team:
☐ Yes
☐ No

Do you think the team members were on an equal knowledge level:
☐ Yes
☐ No

If not: did the knowledge transfer work:
☐ Yes
☐ No

Did the team follow the test first paradigm:
☐ Yes
☐ No

Did the team follow the collective code ownership:
☐ Yes
☐ No

Please rate the frequency of internal meetings during the project:
☐ (5) too many
☐ (4)
☐ (3)
☐ (2)

☐ (1)
☐ (0) too few

Please rate the usefulness of internal meetings during the project:
☐ (5) a must
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) wasted time

Please rate the frequency of the general project meetings (with the shareholders) during the project:
☐ (5) too many
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) too few

Please rate the usefulness of the general project meetings (with the shareholders) during the project:
☐ (5) a must
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) wasted time

Do you think the XP paradigm was appropriate for the intended project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Do you think the XP paradigm was appropriate for the team:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)

☐ (0) poor

Please rate your satisfaction with the other developer's work during the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Rate the enforcement of the established agile methods:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

B.4 Management

How many managers were involved in the project:___

With how many managers did you have contact during your time at the project:___

Where there a clear hierarchy in the management:
☐ Yes
☐ No

Please rate the perceived competence of the project manager:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Did the targets of the project manager match with the upper management/shareholders:
☐ Yes
☐ No

Please rate the agreement of the project manager with the upper management/shareholder:

☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate your satisfaction with the project manager's work during the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate your satisfaction with the upper management/shareholders during the project:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

Please rate the leadership style of the project manager:
☐ (5) excellent
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) poor

How many layers of management did you experience during the project:___

Do you think there was a common project plan at the beginning of the project:
☐ Yes
☐ No

Do you think there was a common project plan during the project:
☐ Yes
☐ No

Do you think there was a common project plan for the end phase of the project:

☐ Yes
☐ No

Please rate the organization of the project:
☐ (5) plan driven
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ (0) chaotic

With your knowledge today do you think this project could be done successfully?:
☐ Yes
☐ No

With the experience made in the project would you participate again?:
☐ Yes
☐ No

B.5 XP Practices in General

The following topics are to be rated by the following scale:
10...fanatic, 9...always, 8...regular, 7...often, 6...usually, 5...half and half
4...common, 3...sometimes, 2...rarely, 1...hardly ever, 0...disagree/never.

How strict/effective did you apply the following...					
Automated Unit Tests		Customer Acceptance Tests		Test First Design	
Pair Programming		Refactoring		Release Planning	
Customer Access		Short Releases		Stand Up Meeting	
Continuous Integration		Coding Standards		Collective Ownership	
Sustainable Pace		Simple Design		System Metaphor	
Lessons Learned		Growth (are team members getting smarter over time?)		Synergy	
Morale		Artifact Reduction (do 'just enough' documentation)			

Table B.2: Strictness of XP practices on a scale between 10 ("fanatic") and 0 ("disagree/never").

B.6 Experience with XP Practices

B.6.1 Planning Game/Release Planning: Plan work incrementally

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.3: Planning game/release planning. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.2 Small Releases: Release as quickly as possible to increase time to market, and get feedback as soon as possible.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.4: Small releases. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.3 Metaphor: If possible, define a metaphor for the system being developed. For example, the shopping cart metaphor is widely used to describe an online ordering system.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.5: Metaphor. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of “easy introduction” of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.4 Simple Design: Use the simplest design that will work for the functionality (user story) being implemented. Do not design for things that may never actually be used.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.6: Simple design. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.5 Testing: Test everything, and try to automate the testing if possible.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.7: Testing. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.6 Refactoring: Instead of designing the entire system up front, design as you go, making improvements as needed. Change the implementation without changing the interface to the functionality, and use automated testing to determine the impact of the refactoring.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.8: Refactoring. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of “easy introduction” of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.7 Pair Programming: Programming in teams of two (or three) allow for a discussion to occur in real-time that addresses requirement, design, testing, and programming concerns.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.9: Pair programming. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.8 Collective Code Ownership: Anyone on the team can make a change to any code at any time.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.10: Collective code ownership. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.9 Continuous Integration: The entire code base is constantly being rebuilt, and retested in an automated fashion.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.11: Continuous integration. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.10 Sustainable Pace: Ideally, team members do not need to work more than 40 hours per week to meet project deadlines. Burning the midnight oil is chunked by management in favor of consistent, predictable, repeatable delivery.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.12: Sustainable pace. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.11 Coding Standards: In order to maximize communication, coding standards are defined by the team, and used to ensure consistent coding practices.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.13: Coding standards. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.12 On-Site Customer: Having constant and direct access to the customer allows the team to work at the fastest possible speed.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.14: On-site customer. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of “easy introduction” of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.13 Daily Standup Meeting

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.15: Daily standup meeting. Please rate on a scale between 5 (“totally true”) and 0 (“strongly disagree”).

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.14 Whole Team: The team functions as a whole. Members are encouraged to be more generalized than specialized. Learning about all technologies and requirements is encouraged.

How long have you worked with and applied this practice:___

This practice was...					
helpful		easy to learn		easy to apply	
enjoyable		already widely used		easy to introduce in a team	

Table B.16: Whole team. Please rate on a scale between 5 ("totally true") and 0 ("strongly disagree").

Please briefly describe the rating of "easy introduction" of this practice:___

Please briefly describe what are the benefits/problems/drawbacks you see:___

B.6.15 Comments

Please, feel free to give comments to the project:___

B.7 Fundamental Project Questions

Topic	At Project-Start	At Project-End
User involvement - 19%		
Do we have the right users?	☐ Yes ☐ No	☐ Yes ☐ No
Are the users involved often and from the beginning?	☐ Yes ☐ No	☐ Yes ☐ No
Is there a quality recording of the users?	☐ Yes ☐ No	☐ Yes ☐ No
Do we alleviate/encourage user involvement?	☐ Yes ☐ No	☐ Yes ☐ No
Do we know what are the users' needs?	☐ Yes ☐ No	☐ Yes ☐ No
Executive management support - 16%		
Do we have the key executives?	☐ Yes ☐ No	☐ Yes ☐ No
Have the key executives financial share in the project results?	☐ Yes ☐ No	☐ Yes ☐ No
Is a failure of the project acceptable?	☐ Yes ☐ No	☐ Yes ☐ No
Is there a well defined project plan?	☐ Yes ☐ No	☐ Yes ☐ No
Has the project team a financial share in the project results?	☐ Yes ☐ No	☐ Yes ☐ No

Table B.17: Fundamental project questions - part 1.

Topic	At Project-Start	At Project-End
Clear statement of requirements - 15%		
Do I have a clear vision?	☐ Yes ☐ No	☐ Yes ☐ No
Do I have a functional analysis?	☐ Yes ☐ No	☐ Yes ☐ No
Do I have a risk analysis?	☐ Yes ☐ No	☐ Yes ☐ No
Do I have a business case?	☐ Yes ☐ No	☐ Yes ☐ No
Is the project progress measurable?	☐ Yes ☐ No	☐ Yes ☐ No
Proper planning - 11%		
Do we have a problem statement?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have a solution statement?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have the appropriate people in the team?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have a well-founded, hard specification?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have achievable milestones?	☐ Yes ☐ No	☐ Yes ☐ No
Realistic expectations - 10%		
Do we have clear expectation specifications?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have prioritized requirements?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have small milestones?	☐ Yes ☐ No	☐ Yes ☐ No
Are we able to react well to changes?	☐ Yes ☐ No	☐ Yes ☐ No
Are we able to practice prototyping?	☐ Yes ☐ No	☐ Yes ☐ No

Table B.18: Fundamental project questions - part 2.

Topic	At Project-Start	At Project-End
Small project milestones - 9%		
Do we apply the 80/20 rule?	☐ Yes ☐ No	☐ Yes ☐ No
Do we follow a TOP-DOWN design?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have time-limits?	☐ Yes ☐ No	☐ Yes ☐ No
Do we use a prototyping tool?	☐ Yes ☐ No	☐ Yes ☐ No
Can we measure the progress?	☐ Yes ☐ No	☐ Yes ☐ No
Competent staff - 8%		
Do I know which competencies are necessary?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have the right people in the team?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have a training program?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have incentives (awards, bonus)?	☐ Yes ☐ No	☐ Yes ☐ No
Does the staff have an overview?	☐ Yes ☐ No	☐ Yes ☐ No
Ownership - 6%		
Do we have a well-defined organization?	☐ Yes ☐ No	☐ Yes ☐ No
Does everyone know their own role?	☐ Yes ☐ No	☐ Yes ☐ No
Are there success awards/bonuses offered?	☐ Yes ☐ No	☐ Yes ☐ No
Is everybody committed to the project?	☐ Yes ☐ No	☐ Yes ☐ No
Do we have well-defined roles?	☐ Yes ☐ No	☐ Yes ☐ No

Table B.19: Fundamental project questions - part 3.

Topic	At Project-Start	At Project-End
Clear visions and objectives - 3%		
Is the vision shared by everyone?	☐ Yes ☐ No	☐ Yes ☐ No
Does the vision fit the company objectives?	☐ Yes ☐ No	☐ Yes ☐ No
Are the objectives achievable?	☐ Yes ☐ No	☐ Yes ☐ No
Are the objectives measurable?	☐ Yes ☐ No	☐ Yes ☐ No
Are there honest checks for meaningfulness?	☐ Yes ☐ No	☐ Yes ☐ No
Hard working, focused staff - 3%		
Is there appeal for success?	☐ Yes ☐ No	☐ Yes ☐ No
Is there concentration on quantifiable deliverables?	☐ Yes ☐ No	☐ Yes ☐ No
Has every team member part-ownership?	☐ Yes ☐ No	☐ Yes ☐ No
Do the team members work well together?	☐ Yes ☐ No	☐ Yes ☐ No
Do the team members trust each other and is trust increasing?	☐ Yes ☐ No	☐ Yes ☐ No

Table B.20: Fundamental project questions - part 4.

Appendix C

Questionnaire for the Empirical Survey concerning Agile Methods in Austrian IT Industry (German)

In this Appendix, the base questions for the survey concerning the use of agile software development methods in the Austrian IT-Industry are described. This survey was conducted in German language, for the English version refer to Appendix D. Since the survey was conducted as a telephone interview, additional questions and answers emerged through the dialog with the interview participants. These questions and answers are not included here but were integrated in the data analysis in Chapter 4.

C.1 Telefonumfrage Agile Softwareentwicklungsmethoden

Firma:
Name und Tel.-Nr. der Kontaktperson:
Interesse an den Umfrageergebnissen:
☐ Ja
☐ Nein

- Sie sind in der Firma tätig als:
  ☐ Manager
  ☐ Entwickler
- Wie viele Mitarbeiter hat die Firma gesamt:
  - Wie viele Mitarbeiter hat der Standort:
  - Wie viele Mitarbeiter hat die Entwicklungsabteilung:
  - Wie viele Mitarbeiter hat Ihr Team:

- Sind Ihnen Agile Softwareentwicklungsmethoden (SWE-Methoden) generell bekannt?
  ☐ Ja
  ☐ Nein
  - Wenn JA (bekannt), unterstützen Sie persönlich den Einsatz dieser in Ihrer Firma/Team?
    ☐ Ja
    ☐ Nein
  - Wenn NEIN (bekannt, nicht unterstützt), warum nicht (bspw. zu aufwändig, nicht effektiv, etc.)? Bitte beschreiben Sie kurz den Grund:
- Würden Sie generell lieber mehr Zugriff auf Informationen zu Agilen SWE-Methoden haben?
  ☐ Ja
  ☐ Nein
- Glauben Sie, dass Agile SWE-Methoden in der Praxis oft eingesetzt werden?
  ☐ Ja
  ☐ Nein
  - Wenn NEIN (nicht eingesetzt), woran könnte das Ihrer Meinung nach liegen:
- Welche Agilen Methoden fallen Ihnen im Stegreif ein, bitte geben Sie ein paar Beispiele:
- Setzen Sie bereits agile Methoden, bzw. haben Sie versucht eine agile Methodik in Ihrer Firma/Team einzusetzen?
  ☐ Ja
  ☐ Nein
  - Wenn JA (bereits eingesetzt), bitte beschreiben Sie kurz Ihre Erfahrung damit (was funktionierte, was nicht):
  - Wenn NEIN (nicht eingesetzt), warum nicht? Bitte beschreiben Sie kurz den Grund:

- Kennen Sie die agile SWE-Methode “Extreme Programming” (XP)?
  ☐ Ja
  ☐ Nein

- Generieren Sie mit Hilfe von Tools Dokumentation aus dem Source-Code?
  ☐ Ja
  ☐ Nein

- Benutzen Sie “Unit-Tests”?
  ☐ Ja
  ☐ Nein

- Verwenden Sie den “Test-First” Ansatz?
  ☐ Ja
  ☐ Nein

- Führen Sie “Continuous Integration” aus bzw. haben Sie eine “Nightly Build” Plattform?
  ☐ Ja
  ☐ Nein

- Kennen Sie die agile Praktik “Pair Programming” (PP)?
  ☐ Ja
  ☐ Nein

  – Wenn NEIN (unbekannt), bitte beschreiben Sie kurz was Sie sich darunter vorstellen könnten:

  – Wenn NEIN (unbekannt), bitte beschreiben Sie kurz wie es z.B. bei der Fehlersuche aussieht?

  – Wenn JA (bekannt), was sind für Sie die Vorteile von “Pair Programming”? Bitte beschreiben Sie diese kurz:

  – Wenn JA (bekannt), haben Sie “Pair Programming” bereits in Ihrem Team ausprobiert?
    ☐ Ja
    ☐ Nein

    ∗ Wenn JA (eingesetzt), bitte beschreiben Sie kurz das Ergebnis (was funktionierte/ was funktionierte nicht):

· Details: Bitte beschreiben Sie kurz in welchen Situationen/Phasen Sie “Pair Programming” eingesetzt haben (z.B.: Prototyping, Design, Architektur, Entwicklung, Fehlersuche):

∗ Wenn NEIN (nicht eingesetzt), warum nicht? Bitte beschreiben Sie kurz den Grund warum Sie “Pair Programming” noch nicht ausprobiert haben:

- Planen Sie “Pair Pogramming” in Zukunft einzusetzen bzw. zu unterstützen?
  ☐ Ja
  ☐ Nein
  - Wenn JA (PP einsetzen), die Praktik “Pair Programming” alleine oder in Verbindung mit anderen Agilen Methoden oder Praktiken (welche?):
  - Wenn NEIN (PP nicht einsetzen), warum nicht? Bitte beschreiben Sie kurz den Grund:
- Planen Sie andere agile Methoden in Zukunft einzusetzen bzw. zu unterstützen?
  ☐ Ja
  ☐ Nein
  - Wenn JA (andere Methoden einsetzen), bitte geben Sie an welche anderen Methoden Sie in Zukunft einsetzen wollen:
  - Wenn NEIN (kein Plan), warum nicht? Bitte beschreiben Sie kurz den Grund
- Termin-Einschätzungen, Liefertreue, bitte bewerten Sie die folgende Aussage:
  “In der Software-Branche werden Termine generell NICHT eingehalten”
  ☐ (1) trifft voll zu
  ☐ (2)
  ☐ (3)
  ☐ (4)
  ☐ (5) trifft gar nicht zu

Werden Termine eingehalten?	**Glaube bzw. Erwartung**	**Erfahrung**
als Auftraggeber (Kunde)	☐ Ja ☐ eher Ja ☐ eher Nein ☐ Nein ☐ k.A.	☐ Ja ☐ eher Ja ☐ eher Nein ☐ Nein ☐ k.A.
als Beauftragter (Verkäufer)	☐ Ja ☐ eher Ja ☐ eher Nein ☐ Nein ☐ k.A.	☐ Ja ☐ eher Ja ☐ eher Nein ☐ Nein ☐ k.A.

Table C.1: Persönliche Bewertung der Liefertreue

Appendix D

Questionnaire for the Empirical Survey concerning Agile Methods in Austrian IT Industry (English)

In this Appendix the base questions for the survey concerning the use of agile software development methods in the Austrian IT-Industry are described. This survey was originally conducted in German language. The German version of the questionnaire can be found in Appendix C. Since the survey was conducted as a telephone interview, additional questions and answers emerged through the dialog with the interview participants. These questions and answers are not included here but were integrated in the data analysis in Chapter 4.

D.1 Telephone survey "Agile Development Methods"

Company:
Name and telephone No. of contact:
Interested in the survey outcome?
☐ Yes
☐ No

- You work in this company as a:
  ☐ Manager
  ☐ Developer
- How many employees has this company:
  - How many employees are at your location:
  - How many employees work at the development department:
  - How many members are in your team:

- Are you generally aware of "agile software development (SWD) methods" ??
  ☐ Yes
  ☐ No
  - If YES (aware of), do you personally support the adoption of these methods in your company or team?
    ☐ Yes
    ☐ No
  - If NO (aware but no support), why not (e.g., too elaborate, not effective, etc.)? Please, briefly describe your reasons:
- Would you like to have better access to more information concerning agile SWD-Methods?
  ☐ Yes
  ☐ No
- Do you think that agile SWD-Methods are often adopted in practice?
  ☐ Yes
  ☐ No
  - If NO (no adoption), in your opinion, what could be reasons for that?
- Which agile SWD-methods do you remember ad hoc? Please name a few examples:
- Do you already apply agile SWD-methods in practice or did you already try to adopt them in your company or team?
  ☐ Yes
  ☐ No
  - If YES (adoption), please, briefly describe your experiences (what did, what did not work):
  - If NO (no adoption), why not? Please, briefly describe your reasons:
- Do you know the agile SWD-method called "Extreme Programming"(XP)?
  ☐ Yes
  ☐ No

- Do you use tools to automatically generate documentation from your source codes?
  ☐ Yes
  ☐ No

- Do you use “Unit-Tests”?
  ☐ Yes
  ☐ No

- Do you apply the “Test-First” paradigm?
  ☐ Yes
  ☐ No

- Do you adhere to “Continuous Integration” or do you have a “Nightly Build” platform?
  ☐ Yes
  ☐ No

- Do you know the agile practice of “Pair Programming” (PP)?
  ☐ Yes
  ☐ No

  – If NO (unknown), please briefly describe what you could imagine by the term “Pair Programming”:

  – If NO (unknown), please, describe how, e.g., finding bugs look like in your team

  – If YES (known), what do think are the advantages of “Pair Programming”? Please, briefly describe them:

  – If YES (known), did you already try out “Pair Programming” in your team?
    ☐ Yes
    ☐ No

    ∗ If YES (tried out), please briefly describe your experiences (what did, what did not work):

      · Details: Please, briefly describe in which situation or phases of the development process you applied “Pair Programming” (e.g., Prototyping, Design, Architecture, Development, Debugging):

- ∗ If NO (not tried out), why not? Please, briefly describe your reasons why you did not try out "Pair Programming":

- Do you have plans for the future to adopt or support the practice of "Pair Programming"?
  ☐ Yes
  ☐ No
  - If YES (plan), are there plans to adopt "Pair Programming" alone or in combination with another agile SWD-Method (which one)?:
  - If NO (no plan to), why not? Please, briefly describe you reasons:

- Do you have plans for the future to adopt or support other agile SWD-Methods in in your team or company?
  ☐ Yes
  ☐ No
  - If YES (adopt other methods), please, briefly describe which methodologies you plan to adopt or support:
  - If NO (no other methods), why not? Please, briefly describe your reasons:

- Dead-Line assessment, delivery reliability, please rate the following phrase:
  "In the software-branch dead-lines are generally NOT met."
  ☐ (1) totally true
  ☐ (2)
  ☐ (3)
  ☐ (4)
  ☐ (5) not true at all

Are dead-lines met?	**Belief / Expectation**	**Experience**
as a sponsor (customer)	☐ Yes ☐ rather Yes ☐ rather No ☐ No ☐ n.s.	☐ Yes ☐ rather Yes ☐ rather No ☐ No ☐ n.s.
as an appointee (supplier)	☐ Yes ☐ rather Yes ☐ rather No ☐ No ☐ n.s.	☐ Yes ☐ rather Yes ☐ rather No ☐ No ☐ n.s.

Table D.1: Personal rating of adherence to deadlines

Appendix E

Scrum Transition - Collected Burn-Down Graphs

In this appendix the collected burn-down graphs of a whole year during an IT-department's transition to scrum, are documented. Sprint relevant information and some statistical data for each sprint are given aside the sprint burn-down graph. Interpretation of the data as well as a discussion of some remarkable sprints, according to the graphs and notes, is given in Chapter 5.

How to read the burn-down graph The vertical bar (shaded green) represents the development power of the team in percent. It is decreased by the absence of team members (vacations and meetings, etc.). The dotted graph (gray) represents the ideal burn-down, which can never be exactly reached due to the granularity of the user stories, but gives a hint where the real graph should converge to. The dashed graph (green) represents the planned, and the solid graph (blue) the real burn-down of the user story points.

E.1 Initial Sprint

Sprint phase: Wednesday 7th January 2009 - Monday 19th January 2009

Sprint 1 Remarks

- The first sprint was one day shorter as the normal sprint due to a national holiday.
- The team members' leave decreased the available person days by 3.
- One team member worked on non sprint related issues.
- The review of two user stories brought up software defects which had impact on the rest of the user stories and their architecture. This led to an overall delay.

- The Product-Owner became sick and his substitute did not have all needed information.
- The estimation meeting for the second sprint was not done during the sprint as planned due to lack of time of the Product-Owner.

Sprint length in days	9
Original number of team members	7
Theoretically available person days	63
Actual available person days	42
Person days available for development	23
Number of user stories	8
Number of all tasks	46
Number of all delayed tasks	5
Number of unplanned tasks	10
Time needed for unplanned tasks [h]	-
User story points committed	32
User story points scored	32

Table E.1: Statistics for sprint 1.

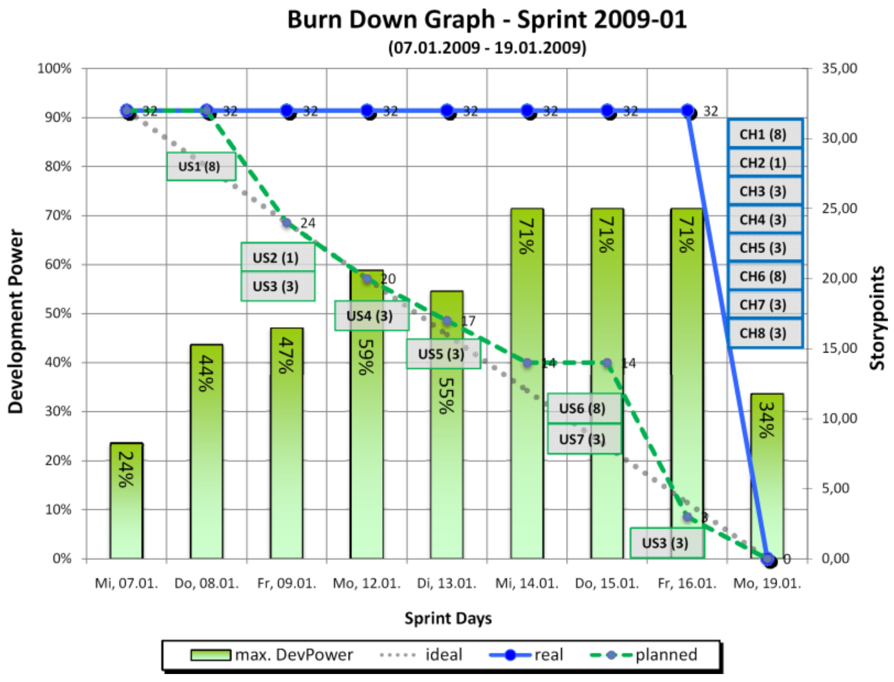


Figure E.1: The burn-down graph of the first sprint.

E.2 Sprint 2

Sprint phase: Tuesday 20th January 2009 - Monday 2nd February 2009

Sprint 2 Remarks

- During the second week of the sprint the Scrum-Master was on leave and had to be substituted.
- The team members' leave decreased the available person days by 4.
- One team member worked on non sprint related issues.
- One big user story was not split and proved to be far more complex than in the sprint planning which in combination with the unplanned tasks had its impact on the scored user story points for this sprint.
- 10 unplanned tasks with a total development effort of one person week had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	55.5
Person days available for development	38
Number of user stories	5
Number of all tasks	71
Number of all delayed tasks	35
Number of unplanned tasks	9
Time needed for unplanned tasks [h]	32.5
User story points committed	29
User story points scored	18

Table E.2: Statistics for sprint 2.

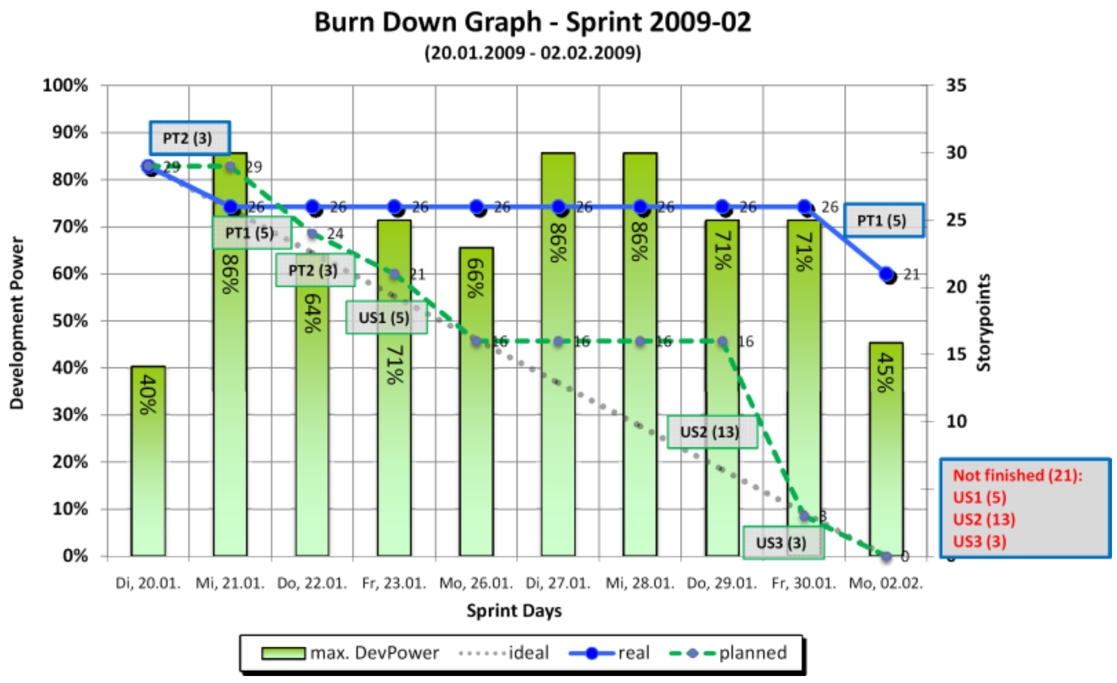


Figure E.2: The burn-down graph of sprint 2.

E.3 Sprint 3

Sprint phase: Tuesday 3rd February 2009 - Monday 16th February 2009

Sprint 3 Remarks

- The team's estimation of the user stories and the commitment was lower than in sprint 2.
- During the first week of the sprint, the Scrum-Master was on leave and had to be substituted.
- There was too much work in parallel. The team did not focus together on one user story.
- One team member worked two days on non sprint related issues.
- Unfinished user stories from the second sprint had to be finished.
- 23 unplanned tasks with a total development effort of 1.3 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	66
Person days available for development	55
Number of user stories	5
Number of all tasks	84
Number of all delayed tasks	39
Number of unplanned tasks	23
Time needed for unplanned tasks [h]	56h
User story points committed	29 (19)
User story points scored	29

Table E.3: Statistics for sprint 3.

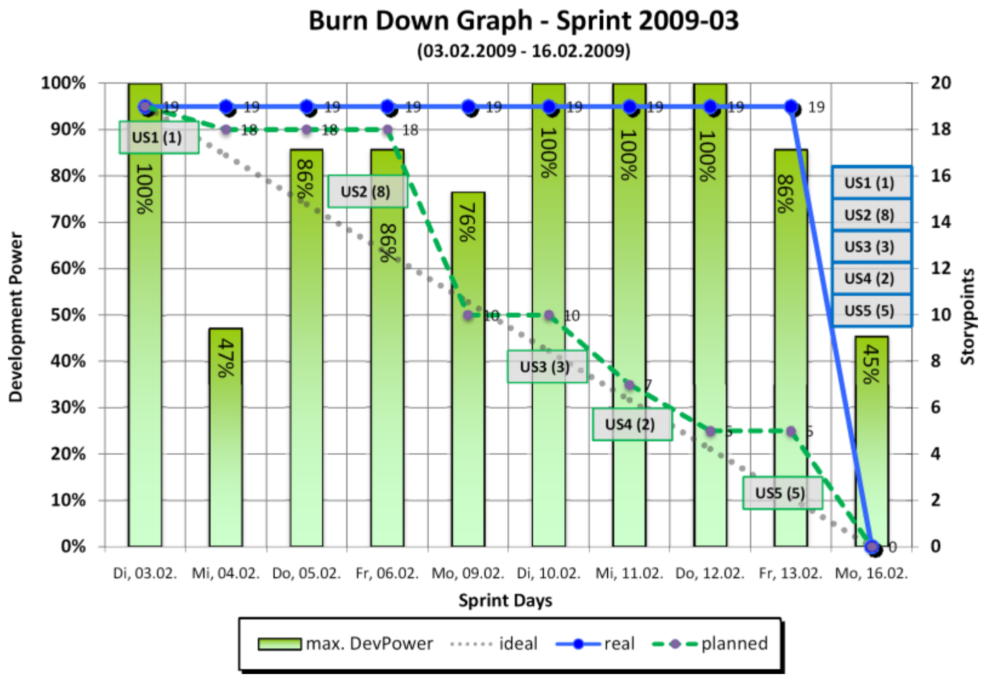


Figure E.3: The burn-down graph of sprint 3.

E.4 Sprint 4

Sprint phase: Tuesday 17th February 2009 - Monday 2nd March 2009

Sprint 4 Remarks

- All the sprint's user stories were related to refactoring. The required refactoring effort were hard to estimate without exact knowledge of the code.
- Only a basic approach to implement the refactoring user stories could be agreed upon in the sprint planning meeting. Therefore the exact procedure and the tasks were partially defined on demand or while doing pair programming respectively.
- The team members' leave decreased the available person days by 9.
- One team member worked on non sprint related issues which reduced the available person days by another 10.
- Tasks and bug fixes from the previous sprint disturbed the current sprint.
- The team started to work together on the user stories sequentially.
- 8 unplanned tasks with a total development effort of 1.6 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	52
Person days available for development	35
Number of user stories	8
Number of all tasks	32
Number of all delayed tasks	12
Number of unplanned tasks	5
Time needed for unplanned tasks [h]	64.25
User story points committed	25
User story points scored	25

Table E.4: Statistics for sprint 4.

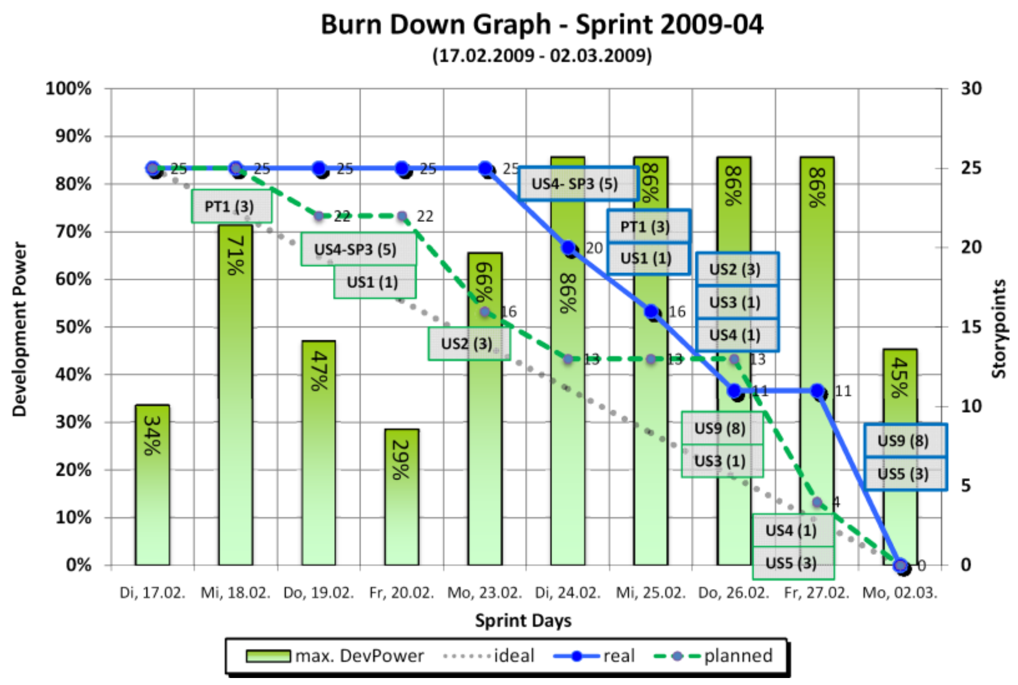


Figure E.4: The burn-down graph of sprint 4.

E.5 Sprint 5

Sprint phase: Tuesday 3rd March 2009 - Monday 16th March 2009

Sprint 5 Remarks

- The team members' leave decreased the available person days by 6.
- One user story turned out to be a never-ending story with many dependencies and hidden defects.
- The team, the Scrum-Master as well as the sprint's Product-Owner were not happy with the amount of unplanned defect fixing tasks the team had been assigned. The Scrum-Master and the Product-Owner organized a meeting with the head of the department concerning this issue.
- 15 unplanned tasks with a total development effort of 1.33 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	65
Person days available for development	52
Number of user stories	6
Number of all tasks	50
Number of all delayed tasks	12
Number of unplanned tasks	15
Time needed for unplanned tasks [h]	51.25
User story points committed	16
User story points scored	16

Table E.5: Statistics for sprint 5.

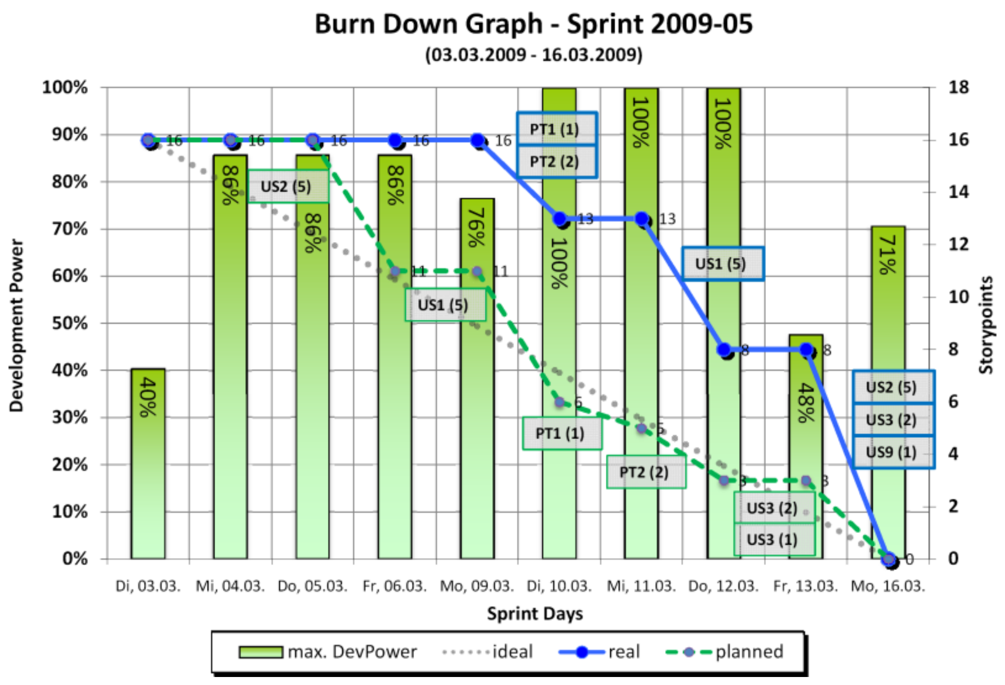


Figure E.5: The burn-down graph of sprint 5.

E.6 Sprint 6

Sprint phase: Tuesday 17th March 2009 - Monday 6th April 2009

Sprint 6 Remarks

- This sprint lasted three weeks.
- The team members' leave decreased the available person days by 5.
- The Scrum-Master was five days on leave in the last week before sprint review.
- The Product-Owner forgot two user stories which were really important to the customer - so there had to be a second planning and a revision of the commitment.
- The Product-Owner reported requirement changes during the sprint.
- In spite of full utilization of the team, an extra user story was added which had to be finished because a manager "made a promise" to a customer.
- 14 unplanned tickets with a total development effort of 1.1 person weeks had to be done additionally.

Sprint length in days	15
Original number of team members	7
Theoretically available person days	105
Actual available person days	98
Person days available for development	85
Number of user stories	17
Number of all tasks	96
Number of all delayed tasks	36
Number of unplanned tasks	14
Time needed for unplanned tasks [h]	42.75
User story points committed	31
User story points scored	31

Table E.6: Statistics for sprint 6.

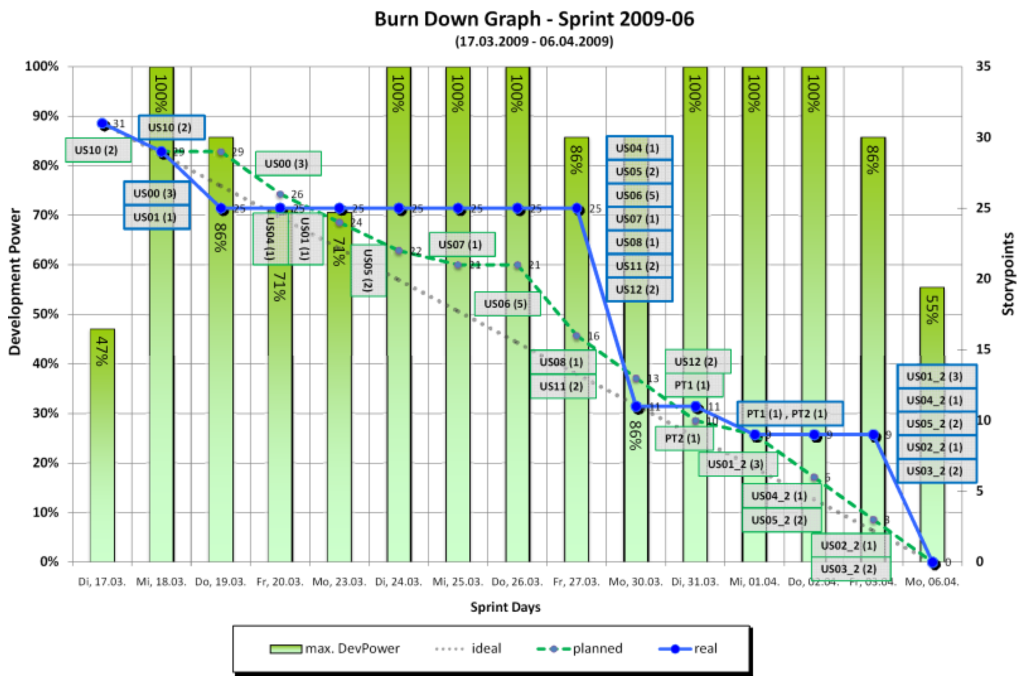


Figure E.6: The burn-down graph of sprint 6.

E.7 Sprint 7

Sprint phase: Tuesday 7th April 2009 - Monday 20th April 2009

Sprint 7 Remarks

- A couple of changes and defects of the previous sprint had a negative impact on this sprint.
- The team members' leave decreased the available person days by 10.
- In this sprint the problem of a single point of knowledge became visible: Knowledge was not distributed through the team. The expert wanted to do all tasks for himself and failed, while the other team members fixed the bugs of the previous sprint. In retrospective they agreed to increase the teamwork and knowledge transfer.
- The Product-Owner reported changes of the requirements during the sprint.
- 10 unplanned tasks with a total development effort of 1.1 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	59
Person days available for development	49
Number of user stories	8
Number of all tasks	41
Number of all delayed tasks	16
Number of unplanned tasks	10
Time needed for unplanned tasks [h]	44.25
User story points committed	20
User story points scored	8

Table E.7: Statistics for sprint 7.

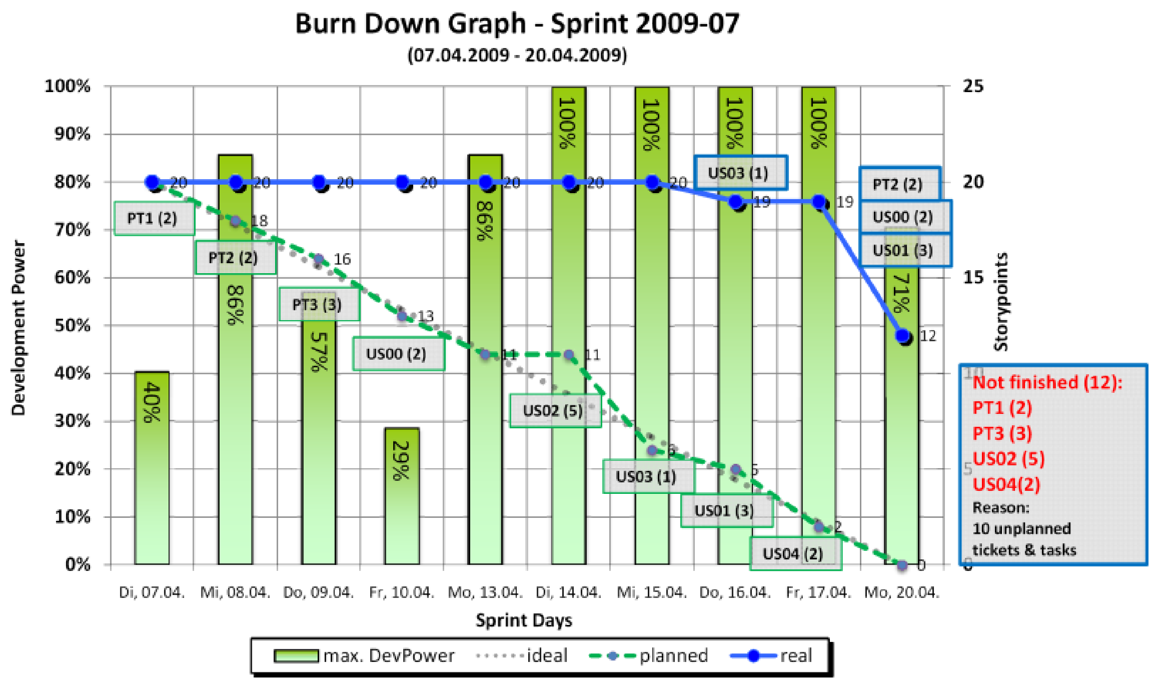


Figure E.7: The burn-down graph of sprint 7.

E.8 Sprint 8

Sprint phase: Tuesday 21st March 2009 - Monday 4th May 2009

Sprint 8 Remarks

- The sprint was one day shorter due to a national holiday.
- The team members' leave decreased the available person days by 4.
- The team committed too much for such a short sprint.
- One team member left the company and did not finish committed tasks.
- An impediment with the testing machine kept the whole team off development for a total amount of 15 person hours. The defect could be found with the help of all team members and the database administrator.
- Due to changes in the requirements which were communicated after the sprint review a lot of functionality became obsolete.
- Communication problems between Product Owners of different projects became a problem. A sprint-external Product-Owner negatively influenced this and previous sprints through directly asking team members to fix small defects of previous

projects. The Scrum-Master as well as the current Product Owner of this sprint were not informed but realized the not sprint related work at the next daily scrum.

- For a scheduled defect fixing task, the defect turned out to be irreproducible. This wasted valuable development time.
- 9 unplanned tasks with a total development effort of 1.8 person weeks had to be done additionally.

Sprint length in days	9
Original number of team members	7
Theoretically available person days	63
Actual available person days	56
Person days available for development	43
Number of user stories	6
Number of all tasks	42
Number of all delayed tasks	26
Number of unplanned tasks	9
Time needed for unplanned tasks [h]	69.8
User story points committed	26
User story points scored	13

Table E.8: Statistics for sprint 8.

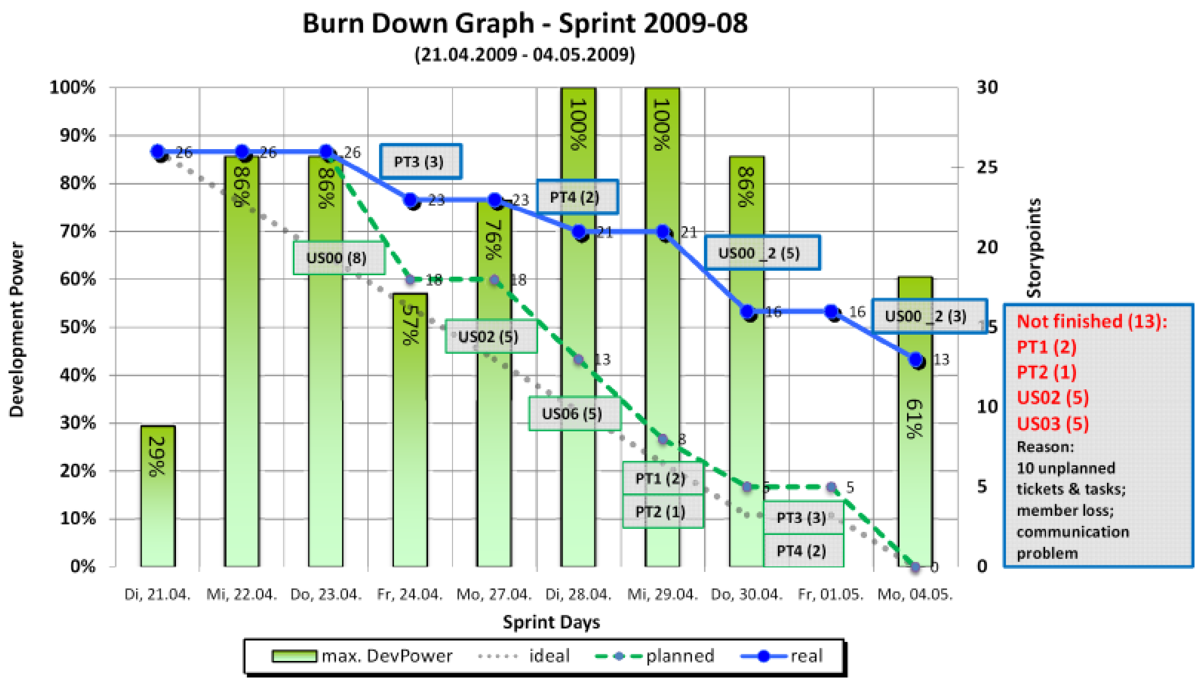


Figure E.8: The burn-down graph of sprint 8.

E.9 Sprint 9

Sprint phase: Tuesday 5th May 2009 - Monday 18th (25th) May 2009

Sprint 9 Remarks

- The sprint 9 was initially planned as a three week sprint. Due to reduction of overtime and leave days the Product-Owner and the team agreed to run a two week sprint and go on leave the third week making a 9-day sprint.
- The team members' leave decreased the available person days for that shorter sprint by 5 more person days.
- One team member left the company and reduced the team at the end of the sprint.
- Unclear requirements of three user stories let the Product-Owner descope them and defer them to a later sprint.
- A sprint-external Product-Owner negatively influenced this sprint through directly asking team members to fix small defects of previous projects. The Scrum-Master as well as the current Product Owner of this sprint were not informed but realized the not sprint related work at the next daily scrum.

- For a scheduled defect fixing task, the defect turned out to be irreproducible. This wasted valuable development time.
- Due to requirement changes the team needed to work until the end of the last sprint day which deferred the overall sprint review.
- 12 unplanned tasks with a total development effort of 0.9 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	7
Theoretically available person days	70
Actual available person days	49
Person days available for development	36
Number of user stories	6 (9)
Number of all tasks	40
Number of all delayed tasks	14
Number of unplanned tasks	12
Time needed for unplanned tasks [h]	32
User story points committed	19 (24)
User story points scored	16

Table E.9: Statistics for sprint 9.

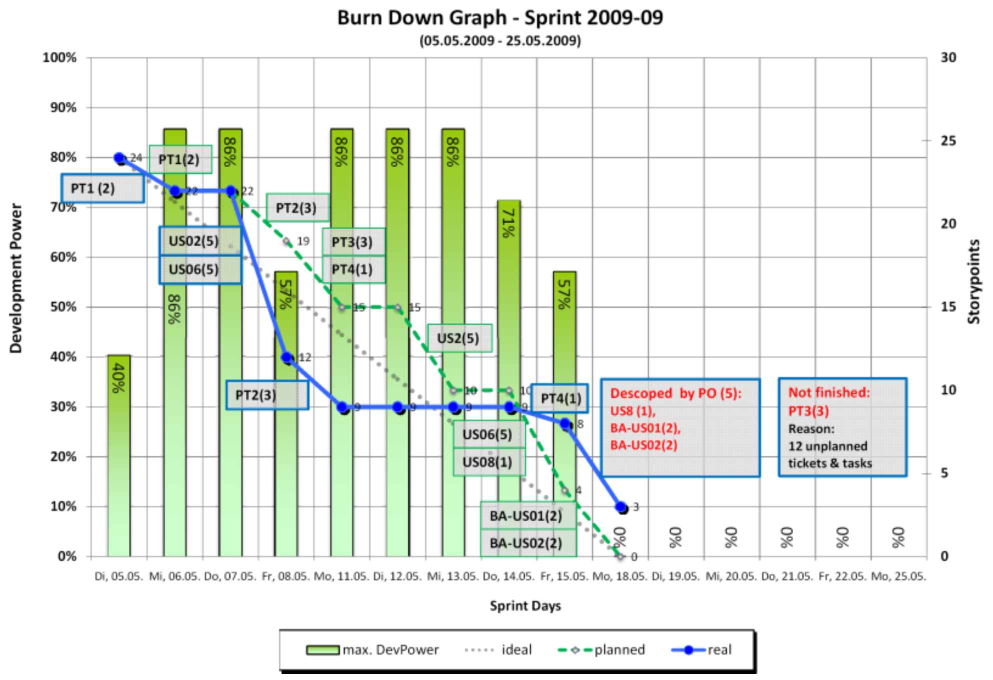


Figure E.9: The burn-down graph of sprint 9.

E.10 Sprint 10

Sprint phase: Tuesday 26th May 2009 - Monday 8th June 2009

Sprint 10 Remarks

- The sprint was one day shorter due to a national holiday.
- The team members' leave decreased the available person days by 1.
- The department-wide scrum team restructuring caused uncertainty in the teams. This was intensified by the fact that two team members had left the company in the previous two sprints.
- One user story had to be descoped by the Product Owner due to lack of information.
- One user story had to be completely canceled by the Product Owner since the customer did not want its functionality any more.
- One planned defect fixing task with low priority was rendered out of scope by the unplanned tasks and could not be finished in this sprint.
- 10 unplanned tasks with a total development effort of 1.3 person weeks had to be done additionally.

Sprint length in days	9
Original number of team members	5
Theoretically available person days	45
Actual available person days	41
Person days available for development	34
Number of user stories	7 (10)
Number of all tasks	34
Number of all delayed tasks	10
Number of unplanned tasks	10
Time needed for unplanned tasks [h]	49.5
User story points committed	12 (18)
User story points scored	12

Table E.10: Statistics for sprint 10.

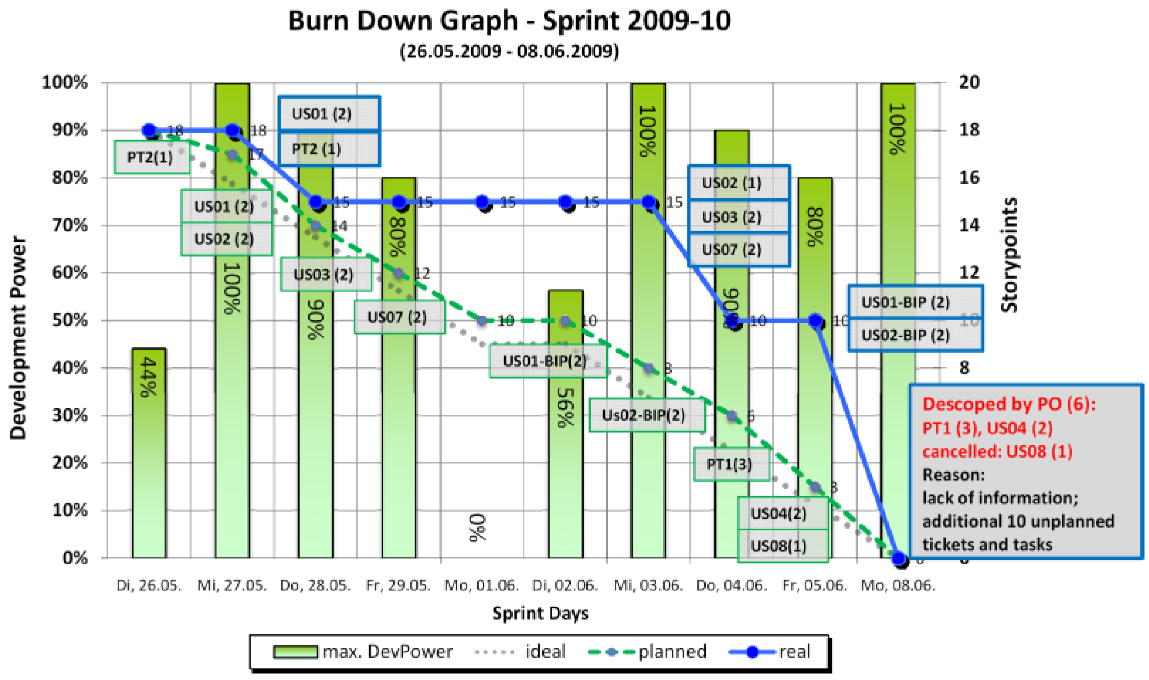


Figure E.10: The burn-down graph of sprint 10.

E.11 Sprint 11

Sprint phase: Tuesday 9th June 2009 - Monday 22nd June 2009

Sprint 11 Remarks

- The sprint was shorter by two days due to national holidays.
- The team members' leave decreased the available person days by 1.
- The team was restructured: two members were exchanged to distribute knowledge between teams.
- The team emphasized design and knowledge transfer.
- The scheduled defect fixing task was descoped by Product-Owner since it had low priority and maintenance had no time an staff to care.
- 8 unplanned tasks with a total development effort of 0.7 person weeks had to be done additionally.

Sprint length in days	8
Original number of team members	5
Theoretically available person days	40
Actual available person days	38.5
Person days available for development	32
Number of user stories	4 (5)
Number of all tasks	27
Number of all delayed tasks	8
Number of unplanned tasks	8
Time needed for unplanned tasks [h]	26
User story points committed	15 (18)
User story points scored	15

Table E.11: Statistics for sprint 11.

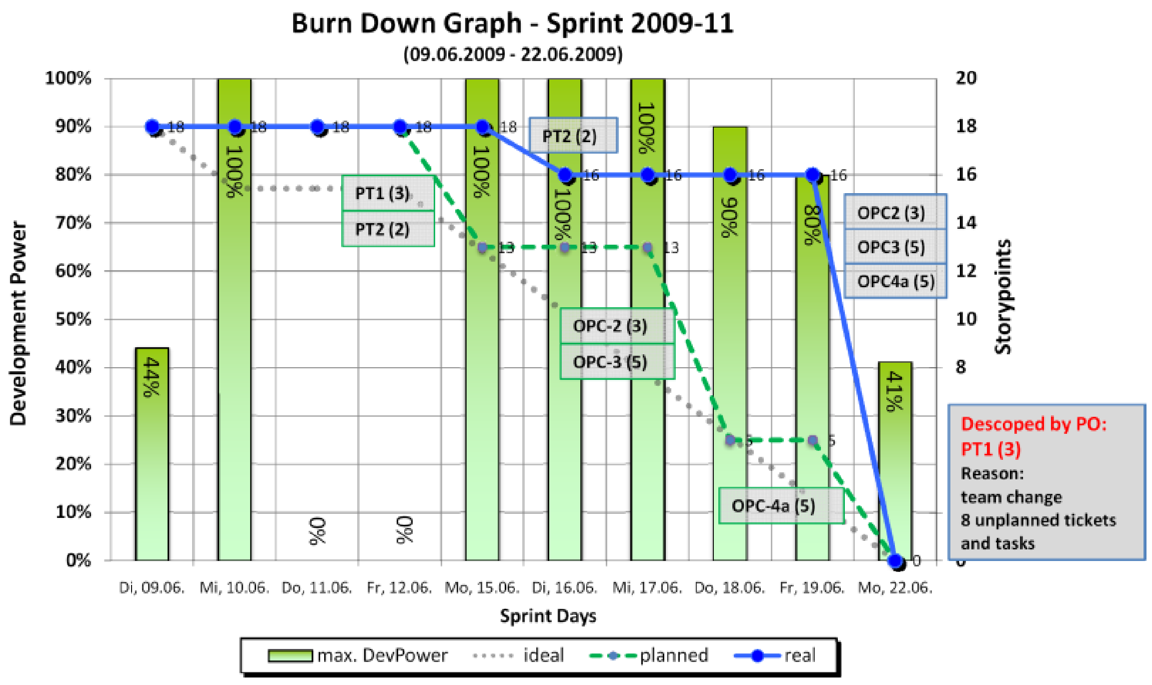


Figure E.11: The burn-down graph sprint 11.

E.12 Sprint 12

Sprint phase: Tuesday 23rd June 2009 - Monday 6th July 2009

Sprint 12 Remarks

- The team members' leave decreased the available person days by 11.
- One team member was half the sprint occupied supporting another team.
- The team accepted an additional user story to be added during the sprint.
- The Product-Owner deferred the sprint review to the next sprint due to lack of time.
- 10 unplanned tasks with a total development effort of 1.6 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	39.5
Person days available for development	28
Number of user stories	6
Number of all tasks	33
Number of all delayed tasks	4
Number of unplanned tasks	10
Time needed for unplanned tasks [h]	63.5
User story points committed	23
User story points scored	23

Table E.12: Statistics for sprint 12.

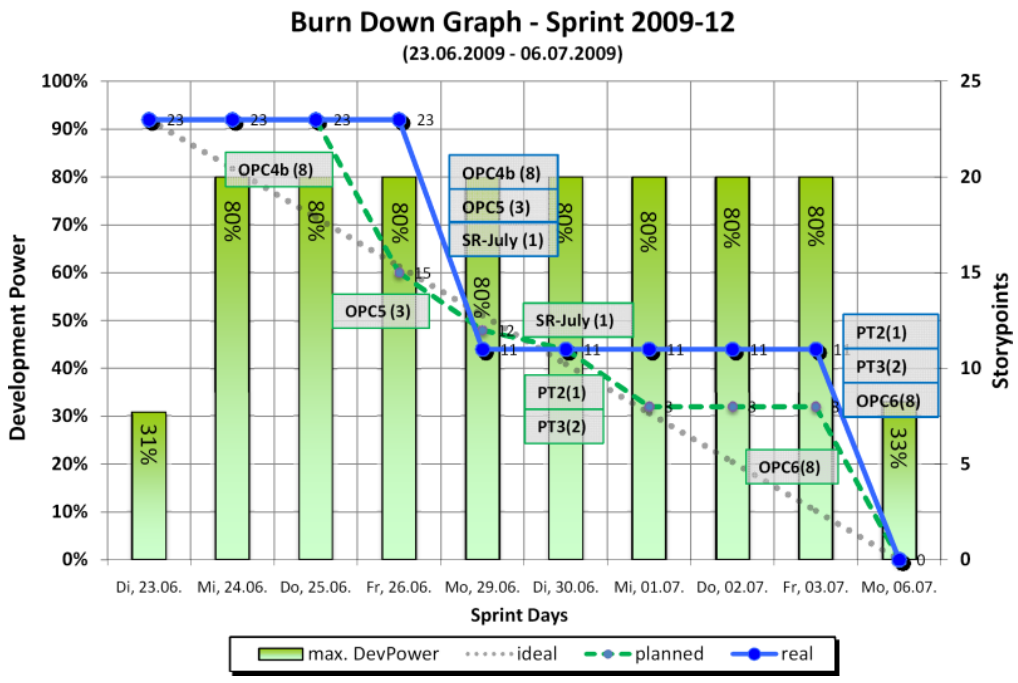


Figure E.12: The burn-down graph of sprint 12.

E.13 Sprint 13

Sprint phase: Tuesday 07th July 2009 - Monday 20th July 2009

Sprint 13 Remarks

- The team members' leave decreased the available person days by 15.
- The Product-Owner reported requirement changes during the sprint which led to a much higher complexity of two user stories, a split and a descoping of two scheduled defect fixing tasks.
- An estimation meeting for the next sprint was done but then the Product-Owner decided that another team should work on this project since it had previously worked in this field. This wasted a lot of time.
- 5 unplanned tasks with a total development effort of 0.5 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	33
Person days available for development	19
Number of user stories	4 (6)
Number of all tasks	36
Number of all delayed tasks	14
Number of unplanned tasks	5
Time needed for unplanned tasks [h]	21
User story points committed	14 (17)
User story points scored	14

Table E.13: Statistics for sprint 13.

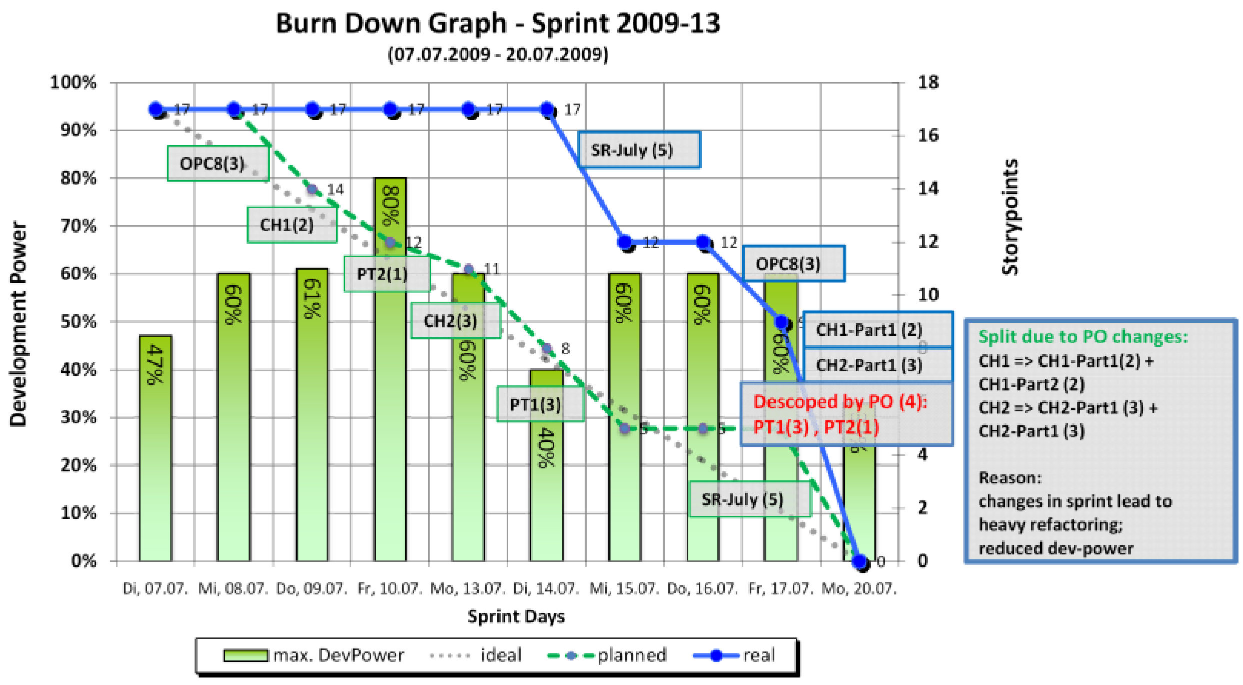


Figure E.13: The burn-down graph of sprint 13.

E.14 Sprint 14

Sprint phase: Tuesday 21st July 2009 - Monday 3rd August 2009

Sprint 14 Remarks

- The Scrum-Master was on leave in the second week of the sprint.
- The team members' leave decreased the available person days by 11.
- There were two Product-Owners in this sprint.
- The Planning meeting for user stories of second Product-Owner took place at the end of the first sprint week.
- Changes in the requirements for the second Product-Owner's user story had negative influence on the development power - therefore two defect fixing tasks with low priorities were descoped.
- 12 unplanned tasks with a total development effort of 1.3 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	27
Person days available for development	39
Number of user stories	6 (8)
Number of all tasks	29
Number of all delayed tasks	10
Number of unplanned tasks	12
Time needed for unplanned tasks [h]	50.5
User story points committed	12 (18)
User story points scored	12

Table E.14: Statistics for sprint 14.

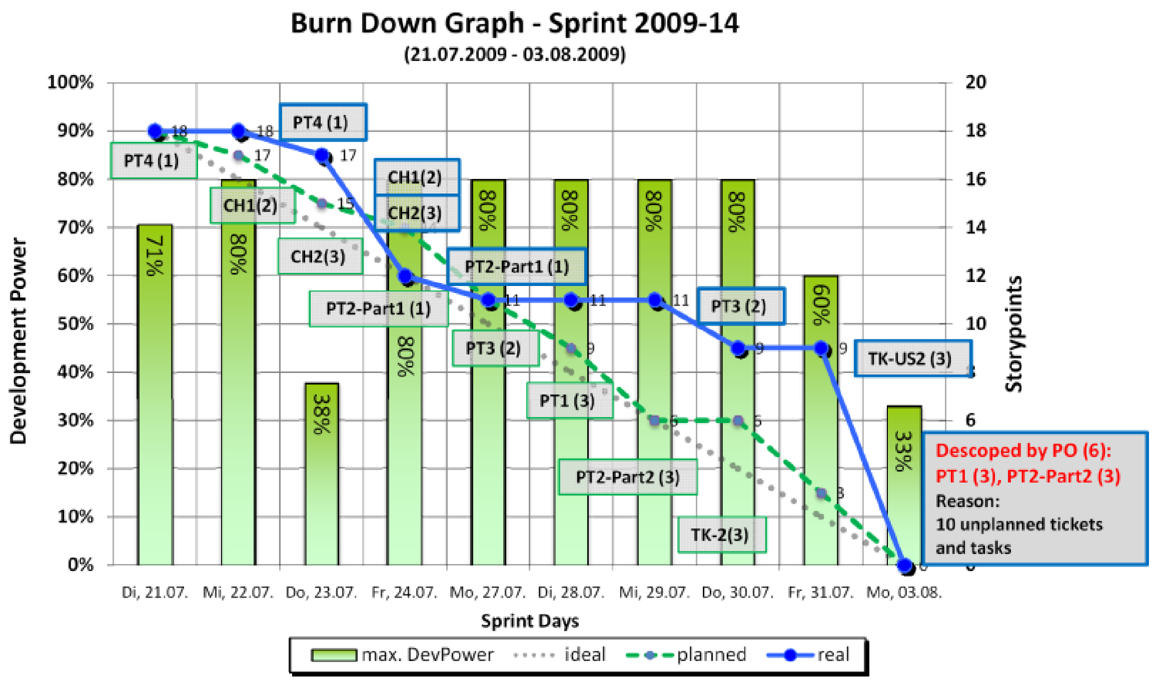


Figure E.14: The burn-down graph of sprint 14.

E.15 Sprint 15

Sprint phase: Tuesday 4th August 2009 - Monday 17th August 2009

Sprint 15 Remarks

- The team members' leave decreased the available person days by 9.
- One user story which did not fit into the last sprint had to be done, although the team was exclusively planned for another Product-Owner's project. This was again the result of communication problem between the Product-Owners.
- The team over-committed in this sprint.
- The last user-story turned out to be too complex and had to be split. The second part could not be finished in this sprint. The project of the second Product-Owner seemed to be in severe time pressure.
- 5 unplanned tasks with a total development effort of 0.4 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	41
Person days available for development	28
Number of user stories	7
Number of all tasks	43
Number of all delayed tasks	19
Number of unplanned tasks	5
Time needed for unplanned tasks [h]	15.5
User story points committed	24
User story points scored	21

Table E.15: Statistics for sprint 15.

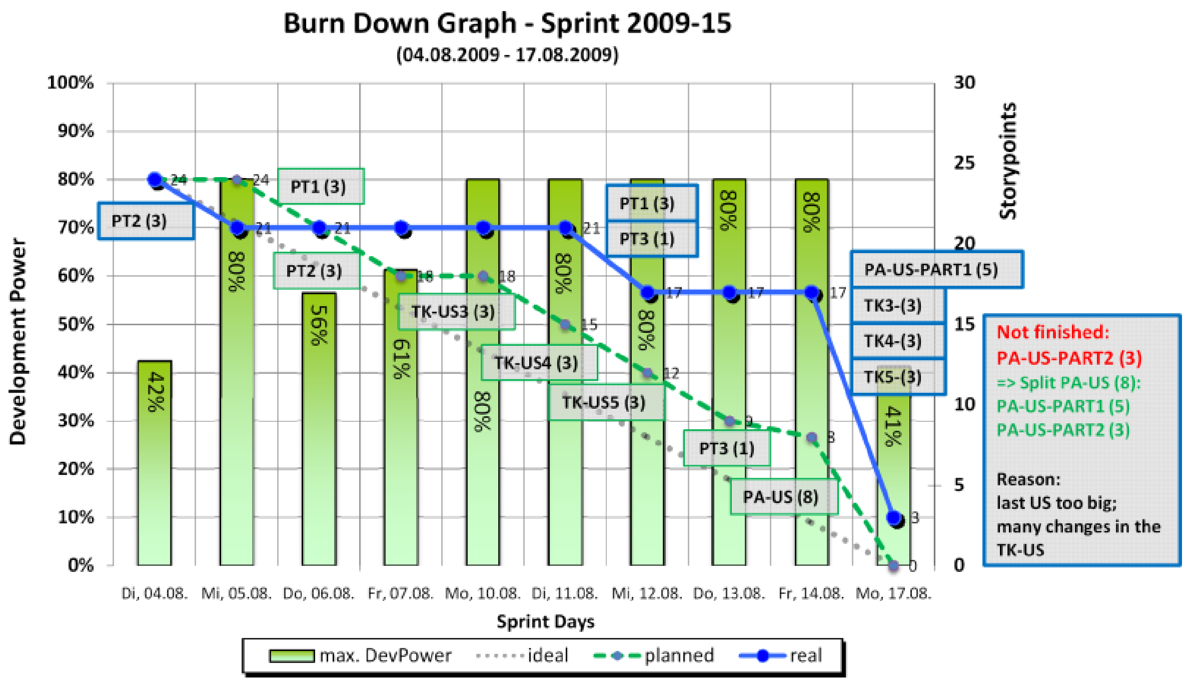


Figure E.15: The burn-down graph of sprint 15.

E.16 Sprint 16

Sprint phase: Tuesday 18th August 2009 - Monday 31st August 2009

Sprint 16 Remarks

- The team members' leave decreased the available person days by 1.
- At the estimation meeting for this sprint the team communicated that it was impossible to implement all the functionality the Product-Owner had planned for this sprint.
- The commitment of the team did not play a role since there was a going-live date.
- The Scrum-Master and the Product-Owner escalated the problem. Unfortunately the department leader was on leave and the substitute did not take the right measurements in spite of the timely communication by the team, the Scrum-Master and the Product-Owner.
- In spite of the overburden of the team it got assigned to one more unplanned task with potential high but unknown priority and to 12 other unplanned defect fixing tasks with high priority.

- These actions had a sustainable negative effect on the team's and Scrum-Master's trust into the management.
- Architecture of this project showed severe shortcomings and a solution could not be found before the middle of the second sprint week. Obviously no architect had checked the functionality before the project was sold and scheduled.
- Together with the Product-Owner, some functionality was descoped and postponed to the following sprint.
- The team power got increased by one developer of another team. The team agreed to come in at the weekend to be able to hold the going-live date.
- One team member deferred his planned holidays for about two weeks. A second team member deferred his planned holiday for one day.
- 13 unplanned tasks with a total development effort of 1.5 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	56
Person days available for development	52
Number of user stories	5
Number of all tasks	82
Number of all delayed tasks	12
Number of unplanned tasks	13
Time needed for unplanned tasks [h]	58.5
User story points committed	32!
User story points scored	32

Table E.16: Statistics for sprint 16.

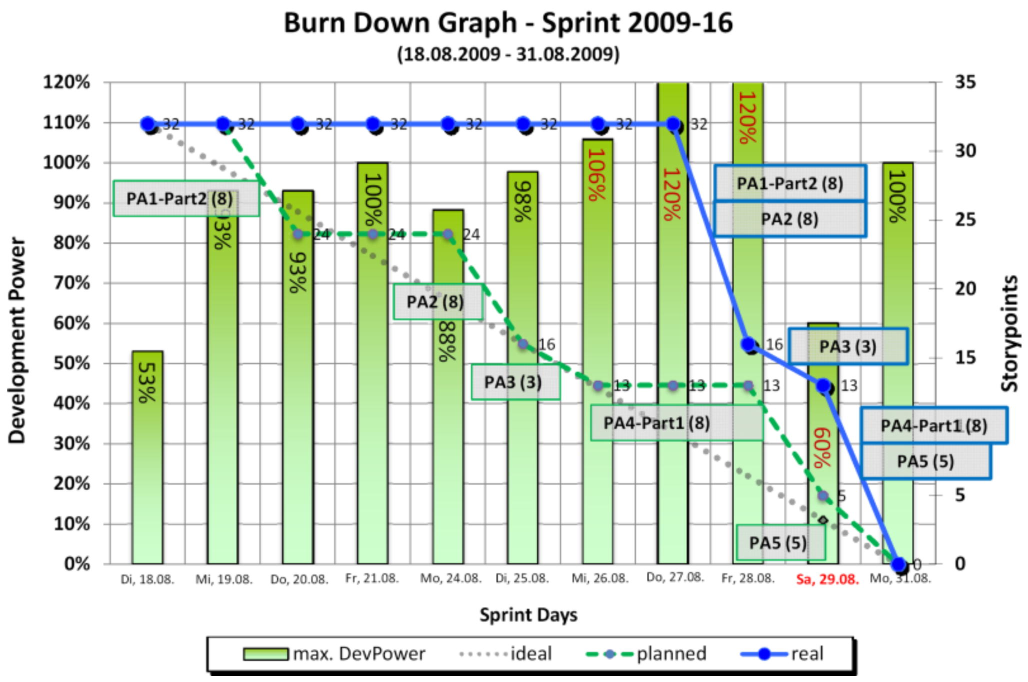


Figure E.16: The burn-down graph of sprint 16.

E.17 Sprint 17

Sprint phase: Tuesday 1st September 2009 - Monday 21st September 2009

Sprint 17 Remarks

- This sprint was a three week sprint.
- The team members' leave decreased the available person days by 15.
- During the last 6 days of the sprint the Scrum-Master on leave and had to be substituted.
- User stories from the last sprint had to be finished for the going live date. The integration phase started in the first week of the sprint. Therefore two Product-Owners shared the team.
- Some planned user stories needed to be clarified by the customer and had to be deferred to the next sprint.
- The team embraced two more defect fixing tasks.
- 5 unplanned tasks with a total development effort of one person day had to be done additionally.

Sprint length in days	15
Original number of team members	5
Theoretically available person days	75
Actual available person days	60
Person days available for development	5
Number of user stories	10 (8)
Number of all tasks	50
Number of all delayed tasks	4
Number of unplanned tasks	5
Time needed for unplanned tasks [h]	8
User story points committed	26
User story points scored	28

Table E.17: Statistics for sprint 17.

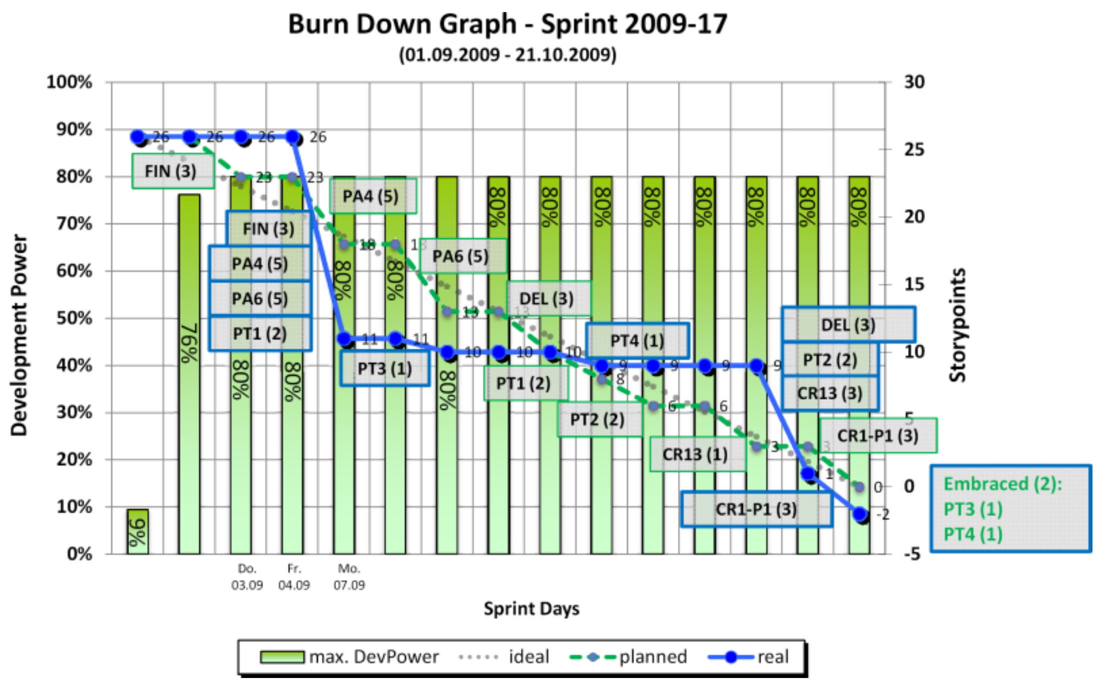


Figure E.17: The burn-down graph of sprint 17.

E.18 Sprint 18

Sprint phase: Tuesday 22nd September 2009 - Monday 5th October 2009

Sprint 18 Remarks

- The Scrum-Master was on leave during the first 4 days of the sprint.
- The team members' leave decreased the available person days by 10.
- The team had to re-estimate 5 of the user stories which showed to be more complex than expected.
- At the sprint planning meeting 4 user stories with an amount of 10 user story points were descoped due to lack of information and requirements. 4 user stories with an amount of 11 user story points were embraced instead.
- Due to the lack of technical information about these user stories, a lot of time was wasted and the completion of the user stories took until the end of the sprint.
- 8 unplanned tasks with a total development effort of 0.9 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	41
Person days available for development	39
Number of user stories	7
Number of all tasks	53
Number of all delayed tasks	11
Number of unplanned tasks	8
Time needed for unplanned tasks [h]	34.5
User story points committed	26
User story points scored	26

Table E.18: Statistics for sprint 18.

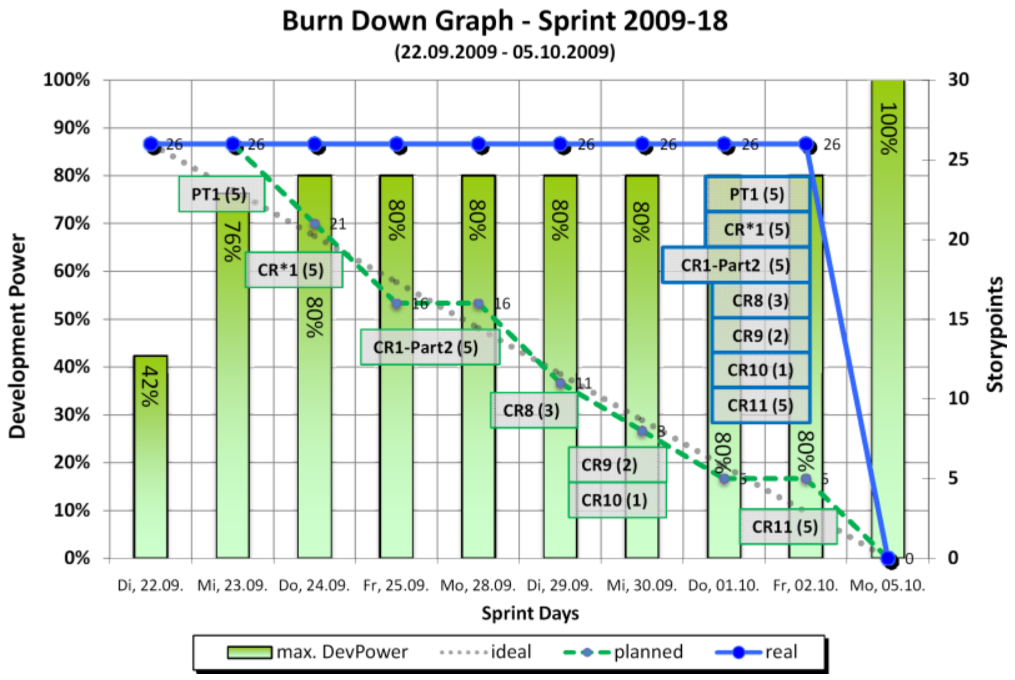


Figure E.18: The burn-down graph of sprint 18.

E.19 Sprint 19

Sprint phase: Tuesday 6th October 2009 - Monday 19th October 2009

Sprint 19 Remarks

- The team members' leave decreased the available person days by 10.
- Due to heavy dependencies between the user stories, most of them were finished at the end of the sprint (see Figure E.19).
- Only a limited number of unplanned tasks occurred.
- Two unplanned tasks with a total development effort of about 0.4 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	41
Person days available for development	39
Number of user stories	14
Number of all tasks	88
Number of all delayed tasks	24
Number of unplanned tasks	2
Time needed for unplanned tasks [h]	14
User story points committed	31
User story points scored	31

Table E.19: Statistics for sprint 19.

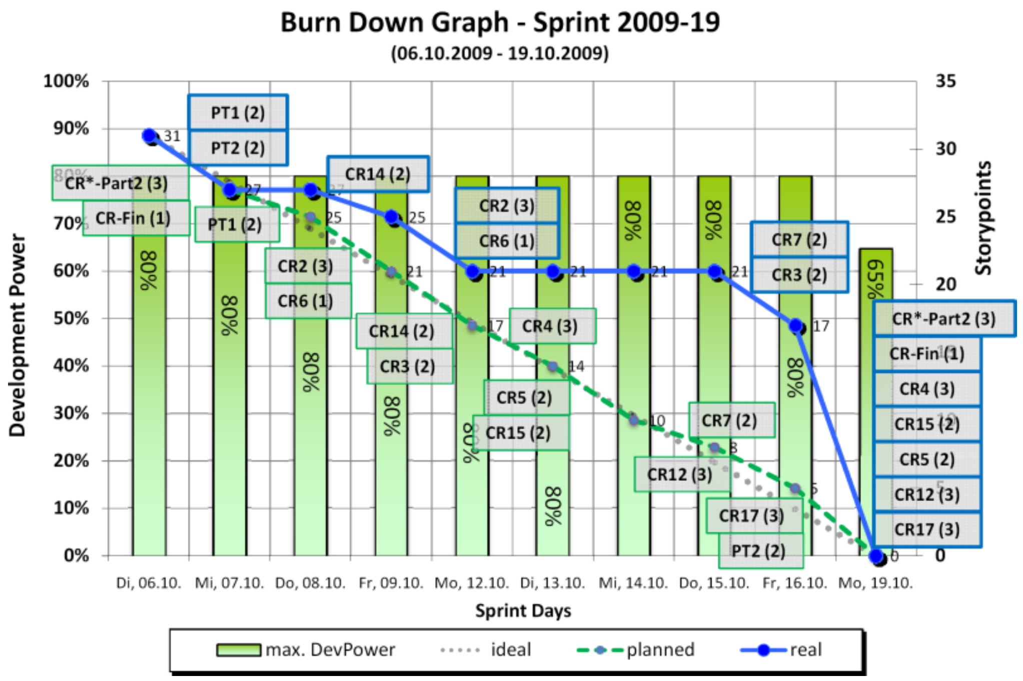


Figure E.19: The burn-down graph of sprint 19.

E.20 Sprint 20

Sprint phase: Tuesday 20th October 2009 - Monday 2nd November 2009

Sprint 20 Remarks

- The sprint was one day shorter as the normal sprint due to a national holiday.
- The team members' leave decreased the available person days by 2.
- The initial plan was to assign 6 different project topics, with 5 different Product-Owners of a total amount of twice the average user story points the team can work off in a normal sprint to the team. The reason was that the different Product-Owners did not communicate and thought they could use the team in this sprint. The Scrum-Master organized a meeting to bring the Product-Owners together to prioritize the projects.
- Support for another team proved less demanding as expected, so another user story with an amount of two user story points was embraced.
- This was the first sprint in which there were no unplanned defect fixing tasks.

Sprint length in days	9
Original number of team members	5
Theoretically available person days	45
Actual available person days	43
Person days available for development	37
Number of user stories	6 (5)
Number of all tasks	34
Number of all delayed tasks	8
Number of unplanned tasks	-
Time needed for unplanned tasks [h]	-
User story points committed	23
User story points scored	25

Table E.20: Statistics for sprint 20.

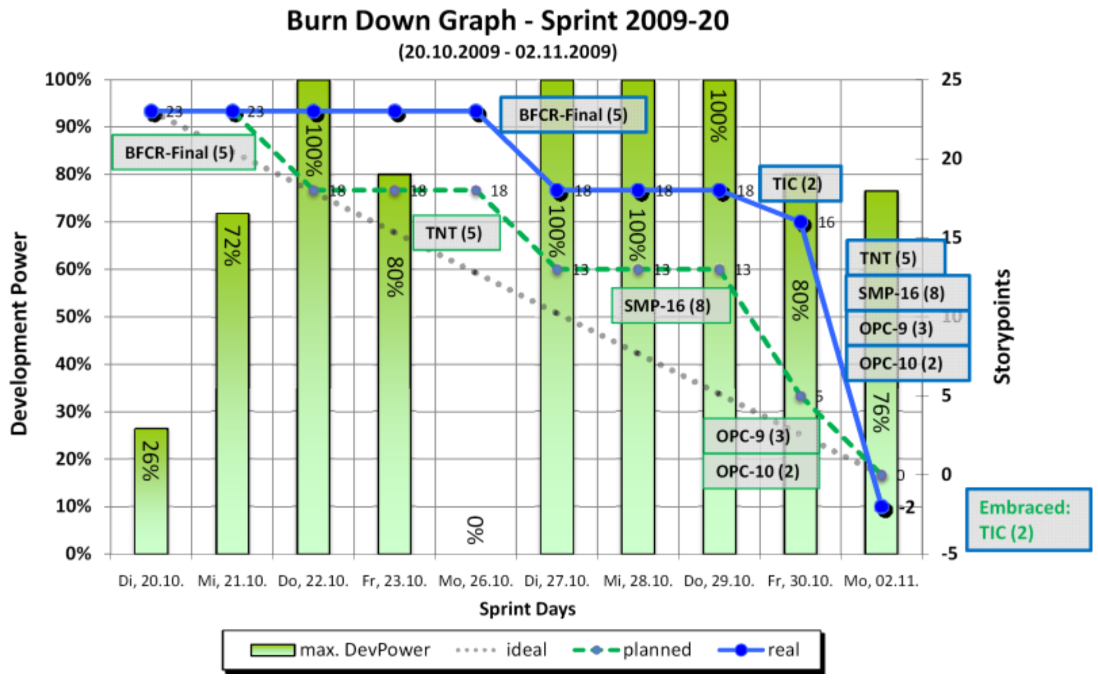


Figure E.20: The burn-down graph of sprint 20.

E.21 Sprint 21

Sprint phase: Tuesday 3rd November 2009 - Monday 16th November 2009

Sprint 21 Remarks

- The team members' leave decreased the available person days by 3.
- Like in the last sprint, too many user stories were initially planned and another meeting was organized with the Product-Owners to prioritize.
- A very detailed and good sprint planning meeting took place. The only drawback was that it happened only one day before sprint begin.
- The sprint estimation and commitment of the team were questioned by the management who feared that the team members would not work 100% and laze around.
- The team became more careful with estimations due to the last sprint experiences (hidden complexities). The team wanted to guarantee their commitment.
- An additional user story was embraced since capacities were free.
- 6 unplanned tasks with a total development effort of about 0.4 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	47
Person days available for development	42
Number of user stories	8 (7)
Number of all tasks	52
Number of all delayed tasks	8
Number of unplanned tasks	6
Time needed for unplanned tasks [h]	15
User story points committed	22
User story points scored	25

Table E.21: Statistics for sprint 21.

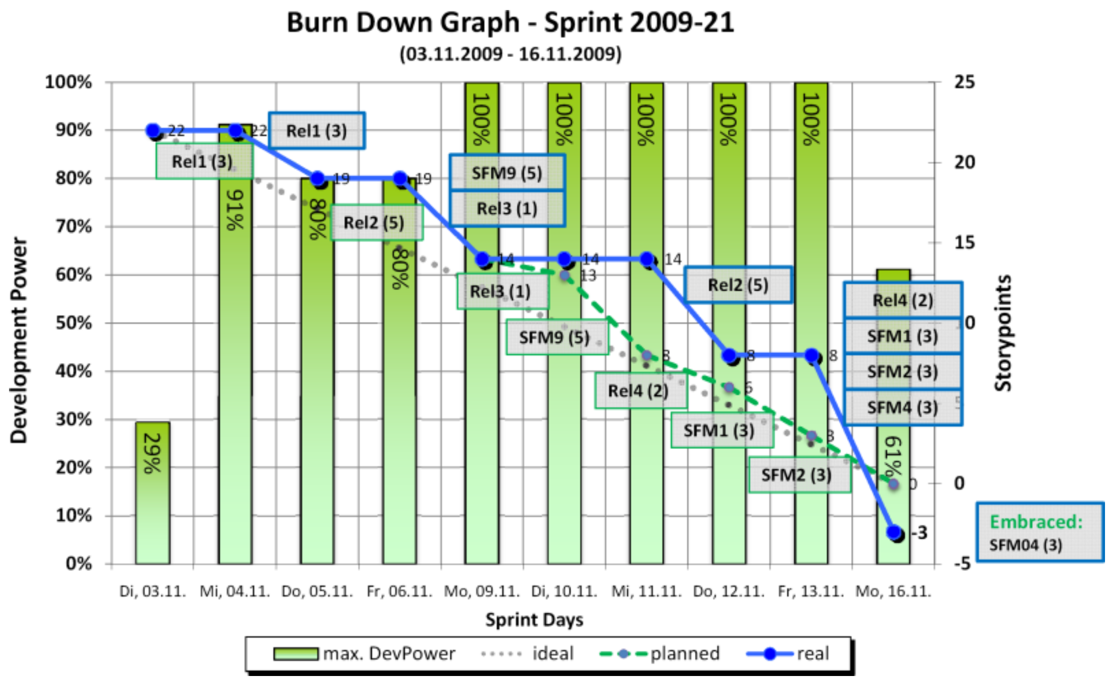


Figure E.21: The burn-down graph of sprint 21.

E.22 Sprint 22

Sprint phase: Tuesday 17th November 2009 - Monday 30th November 2009

Sprint 22 Remarks

- The team members' leave decreased the available person days by 3.
- Management changed the process how defect-fixing tasks were worked off. A certain percentage of the sprint's person days were now reserved for defect fixing tasks. These days are not available for work on the project. The reason was that Product-Owners had frequently descoped low priority defect fixing tasks in previous sprints.
- Careful estimation by the team made it possible to finish critical tasks for integration tests and even embrace another user story.
- 10 unplanned tasks with a total development effort of 1.1 person weeks had to be done additionally.

Sprint length in days	10
Original number of team members	5
Theoretically available person days	50
Actual available person days	43
Person days available for development	37
Number of user stories	6 (4)
Number of all tasks	43
Number of all delayed tasks	6
Number of unplanned tasks	10
Time needed for unplanned tasks [h]	44.5
User story points committed	18
User story points scored	21

Table E.22: Statistics for sprint 22.

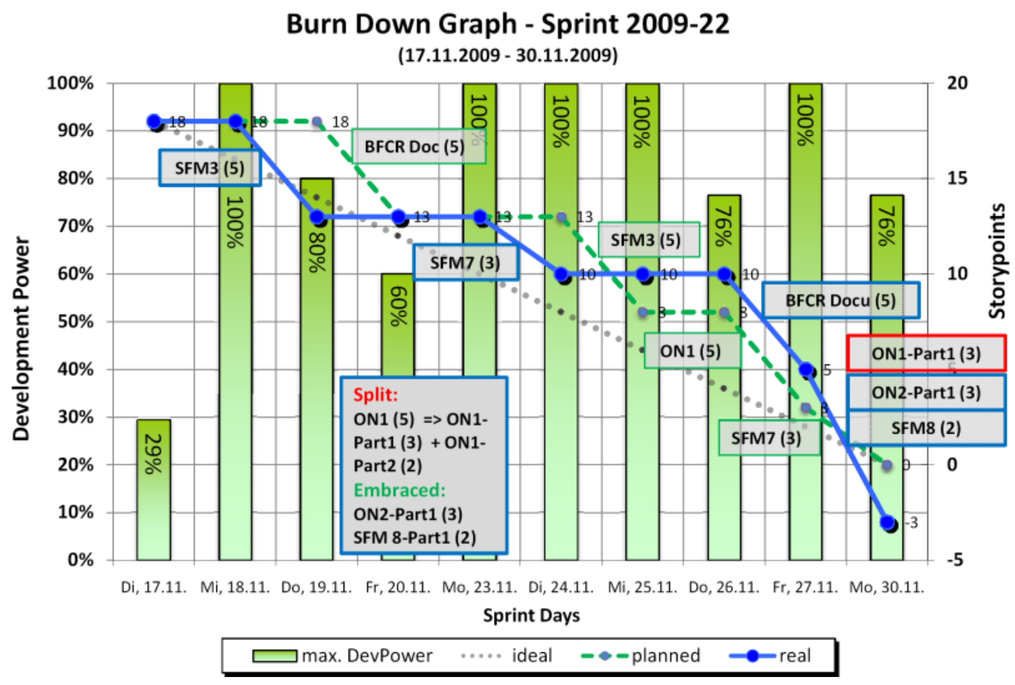


Figure E.22: The burn-down graph of sprint 22.

E.23 Sprint 23

Sprint phase: Tuesday 1st December 2009 - Monday 17th December 2009

Sprint 23 Remarks

- The last sprint of the year was scheduled to be a 12 day sprint.
- The team members' leave decreased the available person days by 4.
- One team member was transferred to another project which decreased the available person days by another 4.
- Architectural changes during the sprint were made necessary through lack of communication between the team and the system architects.
- One unplanned task with a total development effort of 0.2 person weeks had to be done additionally.

Sprint length in days	12
Original number of team members	5
Theoretically available person days	60
Actual available person days	52
Person days available for development	43
Number of user stories	8
Number of all tasks	57
Number of all delayed tasks	7
Number of unplanned tasks	1
Time needed for unplanned tasks [h]	7.5
User story points committed	25
User story points scored	25

Table E.23: Statistics for sprint 23.

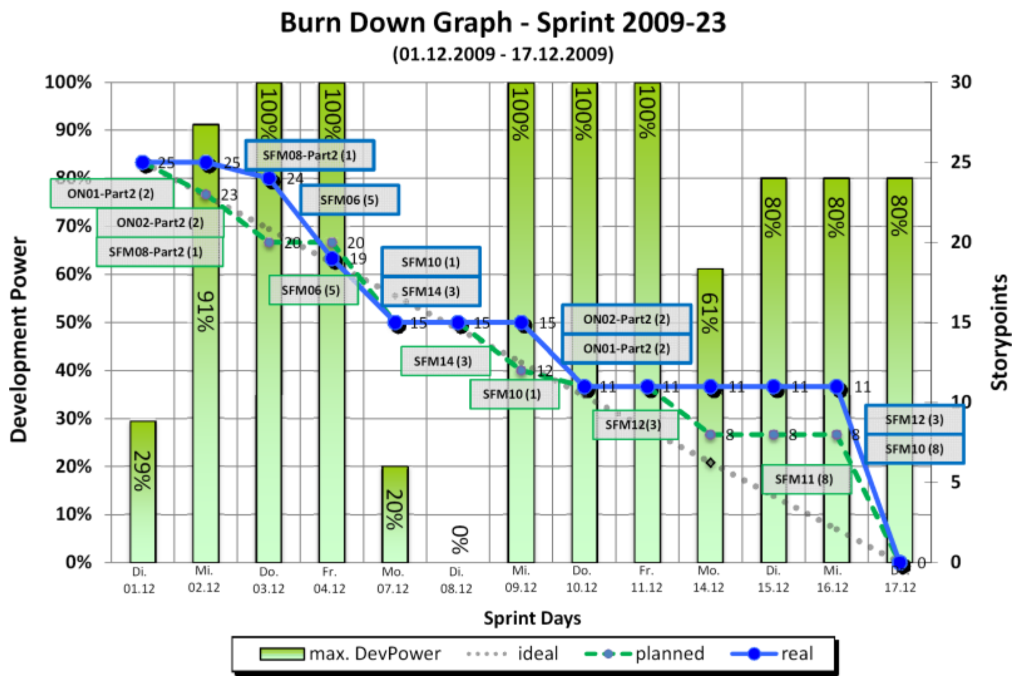


Figure E.23: The burn-down graph of sprint 23.

Appendix F

Scrum Evaluation Questionnaire (German)

In this Appendix the 121 questions for the Scrum evaluation survey are described. This survey was conducted in German language. The questions which could be quantified and their consolidated data are described in Appendix G in English language. Since the evaluation survey was conducted via the web-based tool LimeSurvey, the layout of the questionnaire was different.

F.1 Allgemeine Fragen

Hier werden allgemeine Fragen zur agilen Softwareentwicklung gestellt und die Umstellung der Abteilung auf Scrum.

1 War Dir Scrum bereits vor der Einführung in der Abteilung bekannt? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

2 Wie lange kanntest Du bereits Scrum bevor es in dieser Abteilung eingeführt wurde? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "1" (War Dir Scrum bereits vor der Einführung in dieser Abteilung bekannt?)
Bitte wähle die zutreffende Antwort aus:
☐ länger als 12 Monate
☐ 6-12 Monate
☐ $<$ 6 Monate

3 Hast Du Scrum bereits vor der Einführung praktisch eingesetzt? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

4 Wenn ja, wie lange hast Du damit praktische Erfahrung? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind: Die Antwort war "Ja" bei der Frage "3" (Hast Du Scrum bereits vor der Einführung in dieser Abteilung praktisch eingesetzt?)
Bitte wähle die zutreffende Antwort aus:
☐ länger als 12 Monate
☐ 6-12 Monate
☐ $<$ 6 Monate

5 Sind Dir noch andere agile Softwareentwicklungsmethoden (außer Scrum) bekannt? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

6 Wenn Dir auch andere bekannt sind, bitte zähle diese kurz auf.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "5" (Sind Dir noch andere agile Softwareentwicklungsmethoden (außer Scrum) bekannt?)
Bitte schreibe Deine Antwort hier:...

7 Hast Du diese, Dir bekannten agilen Softwareentwicklungsmethoden (außer Scrum), bereits praktisch eingesetzt? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "5" (Sind Dir noch andere agile Softwareentwicklungsmethoden (außer Scrum) bekannt?)
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

8 Wenn ja, wie lange hast Du damit praktische Erfahrung? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "7" (Hast Du diese, Dir bekannten agilen Softwareentwicklungsmethoden (außer Scrum), bereits praktisch eingesetzt?) Bitte wähle die zutreffende Antwort aus:
☐ länger als 12 Monate
☐ 6-12 Monate
☐ $<$ 6 Monate

9 Wie würdest Du Deine Einstellung zur aktuellen Scrum Praxis einschätzen? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr positiv (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ strikt ablehnend (0)

10 Hat sich an Deiner Arbeitssituation durch die Einführung von Scrum etwas verändert? *
Bitte wähle die zutreffende Antwort aus:
☐ viel besser
☐ eher besser
☐ gleich
☐ eher schlechter
☐ viel schlechter

11 Empfundene Vorteile unserer momentanen Scrum Praxis.
Bitte wähle alle Punkte aus, die zutreffen:
☐ Team-Work
☐ bekannte Sprintlänge
☐ tägliches Feedback
☐ priorisiertes Entwickeln
☐ Selbstorganisation
☐ Sonstiges:

12 Empfundene Nachteile unserer momentanen Scrum Praxis.
Bitte wähle alle Punkte aus, die zutreffen:
☐ Sprint zu kurz
☐ permanente Kontrolle/Überwachung
☐ Änderungen während des Sprints
☐ Keine Möglichkeit für Wissenserwerb außerhalb der Projekte
☐ Sonstiges:

13 Wenn du entscheiden könntest: würdest du wieder zur Arbeitsweise vor Scrum zurückkehren? *
Bitte wähle die zutreffende Antwort aus:
☐ auf jeden Fall bei Scrum bleiben (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ unbedingt zurückkehren (0)

14 Optional, bitte gib an was spricht für a.) die frühere Arbeitsweise und b.) die Beibehaltung von Scrum?
Bitte schreibe Deine Antwort hier:...

15 Wie hast Du die Einführung von Scrum in Deiner Abteilung empfunden? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr positiv (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr negativ (0)

16 War die Information über die Umstellung ausreichend? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

17 Was ist positiv gelaufen?
Bitte wähle alle Punkte aus, die zutreffen:
☐ zeitgerechte Information
☐ transparenter Umstellungsplan
☐ ausreichende Schulungen
☐ individuelle Information
☐ gute Vorbereitung
☐ Sonstiges:

18 Was hätte man verbessern können?
Bitte wähle alle Punkte aus, die zutreffen:
☐ Information früher kommunizieren
☐ Plan der Umstellungsplan transparenter gestalten
☐ mehr Schulungen zu dem Thema abhalten
☐ bessere Vorbereitung
☐ Sonstiges:

19 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.2 Team

Hier werden Fragen zum Scrum-Team gestellt.
20 Bitte gib die Zufriedenheit mit deinem Team an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden
☐ (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

21 Gibt es Probleme im Team? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

22 Wenn es Probleme im Team gibt, welcher Art sind diese? (Bitte beschreibe diese kurz)
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “21 [B02]” (Gibt es Probleme im Team?)
Bitte schreibe Deine Antwort hier:...

23 Wenn es Probleme im Team gibt ... *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “21” (Gibt es Probleme im Team?) Bitte wähle alle Punkte aus, die zutreffen:
☐ ...können diese Probleme intern gelöst werden?
☐ ...bedarf es Hilfe von außerhalb des Teams?

24 Würdest Du von dir aus gerne das Team wechseln? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gerne (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

25 Würdest Du einem, von außerhalb des Teams, angeregten Teamwechsel zustimmen? *
Bitte wähle die zutreffende Antwort aus:

☐ sehr gerne (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

26 Gibt es in Deinem Team Wissensinseln? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

27 Wieviel Aufwand fließt im Team in die Wissensweitergabe (Wissensverbreiterung)? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr viel (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ gar keiner (0)

28 In welchem Rahmen findet die Wissensverbreiterung hauptsächlich statt? *
Bitte wähle die zutreffende Antwort aus:
☐ innerhalb Projektarbeit (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ außerhalb Projektarbeit (0)

29 Bitte gib Deine Zufriedenheit mit der Wissensverbreiterung im Team an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

30 Findest Du die Arbeitsverteilung im Team gerecht? *
Bitte wähle die zutreffende Antwort aus:
☐ vollkommen (5)
☐ (4)

☐ (3)
☐ (2)
☐ (1)
☐ überhaupt nicht (0)

31 Wie findest Du die momentane Größe Deines Teams? *
Bitte wähle die zutreffende Antwort aus:
☐ zu groß
☐ passend
☐ zu klein

32 Was ist Deiner Meinung nach die optimale Teamgröße?
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war NICHT "passend" bei der Frage "31 [B10]" (Wie findest Du die momentane Größe Deines Teams? (Teamgröße))
Bitte schreibe hier Deine Antwort(en):
Minimalgröße=...
Maximalgröße=...
Optimalgröße=...

33 Wie funktioniert die Zusammenarbeit im Team? *
Bitte wähle die zutreffende Antwort aus:
☐ intensive Zusammenarbeit (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ gar nicht, arbeiten getrennt (0)

34 Sollte Deiner Meinung nach das Team ein gemeinsames Büro teilen? *
Bitte wähle die zutreffende Antwort aus:
☐ unbedingt (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ nicht nötig (0)

35 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.3 Scrum Master

Hier werden Fragen zum Scrum Master gestellt.

36 Bitte gib die Zufriedenheit mit Deinem ScrumMaster (SM) an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

37 Ist Dir der Aufgabenbereich/Verantwortungsbereich des (SM) klar? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

38 Wenn der Aufgabenbereich/Verantwortungsbereich des SM unklar ist, beschreibe bitte kurz WAS Dir unklar ist.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Nein” bei der Frage “37 [C02]” (Ist Dir der Aufgabenbereich/Verantwortungsbereich des (SM) klar?)
Bitte schreibe Deine Antwort hier:...

39 Gibt es Probleme mit Deinem SM? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

40 Wenn es Probleme mit Deinem SM gibt, welcher Art sind diese? (Bitte beschreibe diese kurz)
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “39 [C03]” (Gibt es Probleme mit Deinem SM?)
Bitte schreibe Deine Antwort hier:...

41 Wenn es Probleme mit Deinem SM gibt ... *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “39 [C03]” (Gibt es Probleme mit Deinem SM?)
Bitte wähle alle Punkte aus, die zutreffen:
☐ ...können diese Probleme intern gelöst werden?
☐ ...bedarf es Hilfe eines Moderators?

42 Bist Du über die Impediments die der SM löst informiert? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

43 Werden Impediments von Deinem SM zeitgerecht gelöst?
Bitte wähle die zutreffende Antwort aus:
☐ vollkommen (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ überhaupt nicht (0)

44 Wie gut unterstützt der SM die Arbeit des Teams? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

45 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.4 Product Owner

Hier werden Fragen zum Product Owner gestellt.

46 Bitte gib Deine Zufriedenheit mit den Product Ownern (POs) an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

47 Sind dir der Aufgabenbereich/Verantwortungsbereich des PO klar? *

Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

48 Wenn der Aufgabenbereich/Verantwortungsbereich des PO unklar ist, beschreibe bitte kurz WAS Dir unklar ist.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Nein” bei der Frage “47 [D02]” (Sind dir der Aufgabenbereich/Verantwortungsbereich des PO klar?)
Bitte schreibe Deine Antwort hier:...

49 Gibt es Probleme mit POs? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

50 Wenn es Probleme mit dem PO gibt, welcher Art sind diese? (Bitte beschreibe diese kurz)
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “49 [D03]” (Gibt es Probleme mit POs?)
Bitte schreibe Deine Antwort hier:...

51 Wenn es Probleme mit POs gibt ... *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “49 [D03]” (Gibt es Probleme mit POs?)
Bitte wähle alle Punkte aus, die zutreffen:
☐ ...können diese Probleme intern gelöst werden?
☐ ...bedarf es Hilfe eines Moderators?

52 Wie gut unterstützt der PO die Arbeit des Teams? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

53 Bitte gib die Qualität der Planung und die Ausarbeitung der Userstories (US) an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)

☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

54 Werden die US früh genug vorgestellt? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

55 Wann werden in der Regel die User Stories (US) vom PO vorgestellt? *
Bitte wähle die zutreffende Antwort aus:
☐ mind. 1 Woche vor dem Sprint
☐ in der Woche vor dem Sprint
☐ unmittelbar vor dem Sprint
☐ im SP1
☐ direkt im Sprint

56 Finden Änderungen von Anforderungen während eines Sprints statt? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr oft (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

57 Sind diese Änderungen Deiner Meinung nach berechtigt (d.h. US nicht abnehmbar wenn Änderung nicht durchgeht)? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war NICHT “niemals (0)” bei der Frage “56 [D07]” (Finden Änderungen von Anforderungen während eines Sprints statt? (Häufigkeit)) Bitte wähle die zutreffende Antwort aus:
☐ immer (4)
☐ meistens (3)
☐ öfters (2)
☐ manchmal (1)
☐ nie (0)

58 Findet ein Direktzugriff des PO auf Team Mitglieder in einem Ausmaß statt, dass die im Daily Scrum geplante Arbeit verzögert wird? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja

☐ Nein

59 Wenn ein störender Direktzugriff des PO stattfindet, wie häufig findet dieser statt? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "58 [D08]" (Findet ein Direktzugriff des PO auf Team Mitglieder in einem Ausmaß statt, dass die im Daily Scrum geplante Arbeit verzögert wird?)
Bitte wähle die zutreffende Antwort aus:
☐ sehr oft (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

60 Findet ein Direktzugriff eines sprintexternen POs auf Team Mitglieder in einem Ausmaß statt, dass die im Daily Scrum geplante Arbeit verzögert wird? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

61 Wenn ein störender Direktzugriff des sprintexternen PO stattfindet, wie häufig findet dieser statt? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "60 [D09]" (Findet ein Direktzugriff eines sprintexternen POs auf Team Mitglieder in einem Ausmaß statt, dass die im Daily Scrum geplante Arbeit verzögert wird?)
Bitte wähle die zutreffende Antwort aus:
☐ sehr oft (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

62 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.5 Meetings

Hier werden Fragen zu den stattfindenden Besprechungen gestellt.

63 Ist Dir der Zweck aller Besprechungen bekannt? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

64 Wenn der Zweck einer Besprechung unklar ist, beschreibe bitte kurz WAS Dir unklar ist.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Nein" bei der Frage "63 [E01]" (Ist Dir der Zweck aller Besprechungen bekannt?)
Bitte schreibe Deine Antwort hier:...

65 Bitte gib Deine Zufriedenheit mit der Durchführung der Besprechungen an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

66 Gibt es Deiner Meinung nach überflüssige Meetings? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

67 Welche Meetings sind Deiner Meinung nach überflüssig?
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "66 [E02a]" (Gibt es Deiner Meinung nach überflüssige Meetings?)
Bitte schreibe Deine Antwort hier:...

68 Wird mit jeder der Besprechungen der vorgesehene Zweck erreicht? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

69 Wenn der Zweck der Besprechungen nicht erreicht wird, bitte beschreibe kurz welche das sind und was dabei schief läuft.

Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Nein" bei der Frage "68 [E03]" (Wird mit jeder der Besprechungen der vorgesehene Zweck erreicht?)
Bitte schreibe Deine Antwort hier:...

70 Wie findest Du die Anzahl der Besprechungen? *
Bitte wähle die zutreffende Antwort aus:
☐ viel zu viele (4)
☐ zu viele (3)
☐ passend (2)
☐ zu wenige (1)
☐ viel zu wenige (0)

71 Wie findest Du die Effizienz der Besprechungen? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr effizient (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
sehr ineffizient (0)

72 Wie gut findest Du den Aufwand für die Besprechungen investiert? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

73 Wie stehst Du zu einem allgemeinen "Public Demo Meeting" wo alle Teams die Ergebnisse ihres letzten Sprints vorstellen? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr positiv (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ strikt ablehnend (0)

74 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.6 Projektabwicklung

Hier werden Fragen zur generellen Projektabwicklung gestellt.

75 Bitte gib die Zufriedenheit mit der generellen Projektabwicklung an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

76 Wäre es für Dich wichtig zu wissen wie Projekte zustande kommen? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

77 Wie gut verträgt sich Deiner Meinung nach die Projektplanung mit unserer Umsetzung von Scrum? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

78 Hast Du den Eindruck, dass genug Zeit für die Planung und Umsetzung der Projekte vorhanden ist? *
Bitte wähle die zutreffende Antwort aus:
mehr als genug (4)
☐ eher mehr als genug (3)
☐ ausreichend (2)
☐ eher zu wenig (1)
☐ zu wenig (0)

79 Wie Häufig gibt es ”forced commitments”, sodass Umfang und zeitlicher Rahmen vom Kunden/PO bereits vorgegeben sind (Teamschätzungen sind irrelevant

oder haben nur informativen Charakter)? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr oft (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

80 Wie findest Du die Dauer eines ”normalen” Sprints von 2 Wochen? *
Bitte wähle die zutreffende Antwort aus:
☐ zu lang (3)
☐ passend (2)
☐ egal (1)
☐ zu kurz (0)

81 Wird mit vollem Einsatz daran gearbeitet (besonders gegen Ende des Sprints), das Commitment einzuhalten? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

82 Wenn nein, bitte beschreibe kurz den Grund?
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Nein” bei der Frage “81 [F07]” (Wird mit vollem Einsatz daran gearbeitet (besonders gegen Ende des Sprints), das Commitment einzuhalten?)
Bitte schreibe Deine Antwort hier:...

83 Wie groß ist Deiner Meinung nach der Einsatz Deines Teams, das Commitment einzuhalten? *
Bitte wähle die zutreffende Antwort aus:
☐ maximal (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ minimal (0)

84 Wird der Architektur in den Projekten genug Aufmerksamkeit geschenkt? *
Bitte wähle die zutreffende Antwort aus:
☐ zu viel
☐ passend
☐ zu wenig

85 Wann finden die Einbindung der Architekten und die Überlegungen zur Architektur statt? *
Bitte wähle die zutreffende Antwort aus:
☐ zu spät
☐ zeitgerecht
☐ zu früh

86 Haben die Architekturüberlegungen vom technischen Meeting Einfluss auf die Team-Schätzungen der User Stories (US)? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

87 Wenn die Architekturüberlegungen Einfluss auf die Schätzungen haben, werden diese ... *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "86 [F10]" (Haben die Architekturüberlegungen vom technischen Meeting Einfluss auf die Team-Schätzungen der User Stories (US)?)
Bitte wähle die zutreffende Antwort aus:
☐ meist nach oben revidiert
☐ unterschiedlich revidiert
☐ meist nach unten revidiert

88 Wie oft ist es vorgekommen, dass ein "Architektur-Veto" die Abnahme einer US verzögert bzw. verhindert hat? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr oft (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ niemals (0)

89 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.7 Tickets

Hier werden Fragen zur Abarbeitung von geplanten und ungeplanten Tickets gestellt.

90 Bitte gib die Zufriedenheit mit der Art der Ticketabarbeitung an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

91 Wie gut passt unsere derzeitige Art der Ticketbearbeitung mit dem Scrum-Entwicklungsprozess zusammen? *
Bitte wähle die zutreffende Antwort aus:
☐ vollkommen (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ überhaupt nicht (0)

92 Würdest Du eine andere Form der Ticket Abarbeitung bevorzugen? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

93 Bitte mache einen kurzen Vorschlag für eine andere Vorgehensweise der Ticketabarbeitung.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Ja" bei der Frage "92 [G03]" (Würdest Du eine andere Form der Ticket Abarbeitung bevorzugen?)
Bitte schreibe Deine Antwort hier:...

94 Findest Du die Aufteilung der Tickets unter den Teams gerecht? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

95 Wenn nicht, bitte beschreibe kurz den Grund.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Nein" bei der Frage "94 [G04]" (Findest Du die Aufteilung der Tickets unter den Teams gerecht?)
Bitte schreibe Deine Antwort hier:...

96 Wie findest Du die Anzahl aller Tickets pro Sprint? *

Bitte wähle die zutreffende Antwort aus:
☐ übermäßig
☐ passend
☐ gering

97 Wenn übermäßig, gefährdet die Anzahl aller Tickets das Sprint-Ziel? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "übermäßig" bei der Frage "96 [G05]" (Wie findest Du die Anzahl aller Tickets pro Sprint? (Anzahl))
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

98 Bitte gib die Zufriedenheit mit der Vorab-Analyse der Tickets durch die "Maintenance" an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

99 Bitte erkläre kurz den Grund bzw. Deinen Verbesserungsvorschlag.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Antwort war gleich oder kleiner als "(2)" bei der Frage "98 [G06]" (Bitte gib die Zufriedenheit mit der Vorab-Analyse der Tickets durch die "Maintenance/OCSO" an. (Zufriedenheit))
Bitte schreibe Deine Antwort hier:...

100 Wie oft wurde ein Sprint-Commitment auf Grund der Ticket-Last gebrochen? *
Bitte wähle die zutreffende Antwort aus:
☐ häufig (3)
☐ öfters (2)
☐ manchmal (1)
☐ nie (0)

101 Werden Deiner Meinung nach die Tickets dem jeweils richtigen Team zugewiesen? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

102 Wenn nein, bitte beschreibe kurz den Grund und Deinen Verbesserungsvorschlag.

Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Nein” bei der Frage “101 [G08]” (Werden Deiner Meinung nach die Tickets dem jeweils richtigen Team zugewiesen?)
Bitte schreibe Deine Antwort hier:...

103 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.8 Support

Hier werden Fragen zur Unterstützung von anderen (Scrum) Teams gestellt.

104 Wie gut funktioniert die Unterstützung zwischen den Teams? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr gut (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr schlecht (0)

105 Wenn die Unterstützung nicht optimal funktioniert, bitte beschreibe kurz den Grund und mach einen Verbesserungsvorschlag.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Antwort war gleich oder kleiner als “(3)” bei der Frage “104 [H01]” (Wie gut funktioniert die Unterstützung zwischen den Teams? (Qualität))
Bitte schreibe Deine Antwort hier:...

106 Gibt es klare Richtlinien wie eine Unterstützung ablaufen soll. *

Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

107 Wenn es keine klaren Richtlinien gibt, wäre es Deiner Meinung nach sinnvoll? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Nein” bei der Frage “106 [H02]” (Gibt es klare Richtlinien wie eine Unterstützung ablaufen soll.)
Bitte wähle nur eine der folgenden Antworten aus:

☐ Ja
☐ Nein

108 Gab es bei der Unterstützung anderer Teams Schwierigkeiten? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

109 Wenn es bei der Unterstützung Schwierigkeiten gab, bitte beschreibe diese kurz und mache einen Verbesserungsvorschlag.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “108 [H02b]” (Gab es bei der Unterstützung anderer Teams Schwierigkeiten?)
Bitte schreibe Deine Antwort hier:...

110 Gab es schon mal die Situation, dass Unterstützung für ein anderes Team den eigenen Sprint beeinträchtigt bzw. gefährdet hat? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

111 Wenn ja, bitte gib die Häufigkeit an. *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “110 [H03]” (Gab es schon mal die Situation, dass Unterstützung für ein anderes Team den eigenen Sprint beeinträchtigt bzw. gefährdet hat?)
Bitte wähle die zutreffende Antwort aus:
☐ häufig (3)
☐ öfters (2)
☐ manchmal (1)
☐ nie (0)

112 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

F.9 Delivery

Hier werden Fragen zur Zusammenarbeit mit dem Delivery Team gestellt.

113 Bitte gib Deine Zufriedenheit mit der Zusammenarbeit mit dem Delivery Team

an. *
Bitte wähle die zutreffende Antwort aus:
☐ sehr zufrieden (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr unzufrieden (0)

114 Sind Dir die Aufgaben des Delivery Teams klar? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

115 Wenn die Aufgaben des Delivery Teams unklar sind, beschreibe bitte kurz WAS Dir unklar ist.
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Nein" bei der Frage "114 [I02]" (Sind Dir die Aufgaben des Delivery Teams klar?)
Bitte schreibe Deine Antwort hier:...

116 Ist Dir der Übergabeprozess der Projekte ans Delivery Team klar? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

117 Wenn nein, wäre Deiner Meinung nach ein klar definierter Übergabeprozess sinnvoll? *
Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war "Nein" bei der Frage "116 [I04]" (Ist Dir der Übergabeprozess der Projekte ans Delivery Team klar?)
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

118 Hat es durch die Unterstützung des Delivery-Teams Beeinträchtigungen des Sprints (Gefährdung des Commitments) gegeben? *
Bitte wähle nur eine der folgenden Antworten aus:
☐ Ja
☐ Nein

119 Wenn es durch die Unterstützung zu Sprint-Beeinträchtigungen kam, wie häufig ist das passiert? *

Beantworte diese Frage nur, wenn folgende Bedingungen erfüllt sind:
Die Antwort war “Ja” bei der Frage “118 [I05]” (Hat es durch die Unterstützung des Delivery-Teams Beeinträchtigungen des Sprints (Gefährdung des Commitments) gegeben?)
Bitte wähle die zutreffende Antwort aus:
☐ häufig
☐ (3)
☐ öfters (2)
☐ manchmal (1)
☐ nie (0)

120 Wie stehst Du zu der Idee, die Abnahme beim Kunden durch das eigene Team durchführen zu lassen? *
Bitte wähle die zutreffende Antwort aus:
☐ sehr positiv (5)
☐ (4)
☐ (3)
☐ (2)
☐ (1)
☐ sehr negativ (0)

121 Optionale abschließende Kommentare oder Ergänzungen zu den obigen Fragen.
Bitte schreibe Deine Antwort hier:...

Vielen Dank für die Beantwortung des Fragebogens ■

Appendix G

Scrum Evaluation Detailed Data (English)

In this appendix the data of the scrum evaluation, which could be quantified are listed in tables. The additional comments and questions which were textually answered are omitted here for brevity but are included in the discussion in Chapter 5.

G.1 General Information

Yes	No
20	12
62.5%	37.5%

Table G.1: Q1: Have you been aware of Scrum before it was introduced in this department?

longer than 12 months	6-12 months	$<$ 6 months
7	4	9

Table G.2: Q2: How long did you know Scrum before it was introduced in this department?

Yes	No
2	30
6.25%	93.75%

Table G.3: Q3:Did you have practical experience with Scrum before it was introduced in this department

longer than 12 months	6-12 months	< 6 months
1	1	0
3.12%	3.12%	0%

Table G.4: Q4: How long did you have practical experience with Scrum before it was introduced in this department?

Yes	No
13	19
40.62%	59.37%

Table G.5: Q5: Do you know other agile software development methodologies (other than Scrum)?

Yes	No
4	9
12.5%	29.12%

Table G.6: Q7: Do you have practical experience with the named agile methodologies

longer than 12 months	6-12 months	< 6 months
1	1	2
3.12%	3.12%	6.25%

Table G.7: Q8: How long did you have practical experience with the named agile methods?

absolutely positive (5)	(4)	(3)	(2)	(1)	absolutely negative (0)	positive	negative
3	14	13	2	0	0	30	2
9.38%	43.75%	40.63%	6.25%	0%	0%	93.75%	6.25%

Table G.8: Q9: How would you rate your attitude towards Scrum?

much better	better	same	worse	much worse
7	16	6	3	0
21.87%	50%	18.75%	9.37%	0%

Table G.9: Q10: Did your work experience change with the introduction of Scrum in this department?

Advantage	#votes	votes [%]
team-work	29	90.62%
team-work	29	90.62%
known sprint length	7	21.87%
daily feedback	22	68.75%
prioritized development	8	25%
self organization	15	46.87%

Table G.10: Q11: Advantages through the current Scrum practice.

Disadvantage	#votes	votes [%]
sprint too short	25	78.12%
permanent stress ”	9	28.12%
permanent observation/supervision	10	31.25%
changes during the sprint	22	68.75%
no possibility for knowledge acquisition aside projects	15	46.87%

Table G.11: Q12: Disadvantages through the current Scrum practice.

keep Scrum by all means (5)	(4)	(3)	(2)	(1)	absolutely switch back (0)	positive	negative
11	10	9	2	0	0	30	2
34.37%	31.25%	28.12%	6.25%	0%	0%	93.75%	6.25%

Table G.12: Q13: Would you change back to the pre-Scrum development method, if you could?

absolutely positive (5)	(4)	(3)	(2)	(1)	absolutely negative (0)	positive	negative
3	14	9	5	1	0	26	6
9.37%	43.75%	28.12%	15.62%	3.12%	0%	81.25%	18.75%

Table G.13: Q15: How did you experience the introduction of Scrum in this department?

Yes	No
27	5
84.37%	15.62%

Table G.14: Q16: Was the information for the transition to Scrum sufficient?

timely information	17	53.12%
transparent transition plan	13	40.62%
sufficient training	9	28.12%
individual information	5	15.62%
well preparation	11	34.37%

Table G.15: Q17: What went well with the Scrum transition?

earlier information communication	8	25%
make the migration plan more transparent	7	21.87%
offer more trainings for this topic	13	40.62%
better preparation	5	15.62%

Table G.16: Q18: What could have been better with the Scrum Transition?

G.2 Team

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
18	8	5	0	1	0	31	1
56.25%	25%	15.62%	0%	3.12%	0%	96.87%	3.12%

Table G.17: Q20: Please rate you satisfaction with your Scrum-team.

Yes	No
5	27
15.63%	84.38%

Table G.18: Q21: Do you have problems with your team?

...can these problems be solved internally?	5	15.62%
...do you need a moderator to solve these problems?	3	9.37%

Table G.19: Q23: If you have problems ...

gladly (5)	(4)	(3)	(2)	(1)	never (0)
0	2	4	3	9	14
0%	6.25%	12.5%	9.37%	28.12%	43.75%

Table G.20: Q24: Would you like to change the team?

gladly (5)	(4)	(3)	(2)	(1)	never (0)
0	3	7	5	11	6
0	9.37%	21.87%	15.62%	34.37%	18.75%

Table G.21: Q25: Would you change the team if you were asked by your superior?

Yes	No
26	6
81.3%	18.8%

Table G.22: Q26: Do you have single points of knowledge in your team?

very much (5)	(4)	(3)	(2)	(1)	nothing (0)
1	7	14	8	2	0
3.12%	21.87%	43.75%	25%	6.25%	0%

Table G.23: Q27: How much effort does the team invest for knowledge distribution?

inside project scope (5)	(4)	(3)	(2)	(1)	outside project scope
17	11	3	1	0	0
17	11	3	1	0	0

Table G.24: Q28: In which scope does your team distribute knowledge?

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
2	11	14	3	2	0	27	5
6.25%	34.37%	43.75%	9.37%	6.25%	0%	84.37%	15.62%

Table G.25: Q29: Please rate your satisfaction with knowledge distribution in you team.

absolutely (5)	(4)	(3)	(2)	(1)	absolutely not (0)	positive	negative
3	21	7	1	0	0	31	1
9.37%	65.62%	21.87%	3.12%	0%	0%	96.9%	3.12%

Table G.26: Q30: Is the distribution of work fair among the members of your team?

too big	adequate	too small
5	26	1
15.6%	81.3%	3.1%

Table G.27: Q31: Please rate the current size of your team.

	Minimum Team Size			Optimum Team Size			Maximum Team Size		
size	3	4	5	4	5	6	5	6	7
#votes	2	1	3	1	3	2	1	3	2
votes [%]	33.3	16.7	50	16.7	50	33.3	16.7	50	33.3

Table G.28: Q32: In your opinion what is the a.)minimum, b.)optimum and c.) maximum team size? (The Percentage is related to the number of people who rated the team size either "too big" or "too small", see Table G.27)

intensive team-work (5)	(4)	(3)	(2)	(1)	no teamwork; work alone (0)	positive	negative
12	17	3	0	0	0	32	0
37.5%	53.12%	9.37%	0%	0%	0%	100%	0%

Table G.29: Q33: Please rate the teamwork in your team.

absolutely (5)	(4)	(3)	(2)	(1)	not necessary (0)	positive	negative
11	9	7	0	0	5	27	5
34.37%	28.12%	21.87%	0%	0%	15.6%	84.4%	15.6%

Table G.30: Q34: In your opinion, should the team be located in the same office?

G.3 Scrum-Master

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
15	12	4	1	0	0	31	1
46.87%	37.5%	12.5%	3.12%	0%	0%	96.9%	3.1%

Table G.31: Q36: Please rate your satisfaction with your Scrum-Master.

Yes	No
26	6
81.3%	18.8%

Table G.32: Q37: Are the Scrum-Master's responsibilities clear to you?

Yes	No
1	31
3.1%	96.9%

Table G.33: Q39: Do you have problems with your Scrum-Master?

...can these problems be solved internally?	1	3.12%
...do you need a moderator to solve these problems?"	1	3.12%

Table G.34: Q41: If you have problems with your Scrum-Master ...

Yes	No
28	4

Table G.35: Q42: Are you informed about the impediments the Scrum-Master has to solve?

absolutely (5)	(4)	(3)	(2)	(1)	absolutely not (0)	positive	negative
13	17	1	1	0	0	31	1%
40.62%	46.87%	3.12%	3.12%	0%	0%	96.9%	3.12%

Table G.36: Q43: Are the impediments solved in time from the moment they occur?

very good (5)	(4)	(3)	(2)	(1)	very bad (0)	positive	negative
18	10	2	2	0	0	30	2
56.25%	31.25%	6.25%	6.25%	0%	0%	93.8%	6.3%

Table G.37: Q44: How well does the Scrum-Master support the work of the team?

G.4 Product-Owner

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
4	11	10	3	4	0	25	7
12.5%	34.37%	31.25%	9.37%	12.5%	0%	78.1%	21.9%

Table G.38: Q46: Please rate you satisfaction with the Product-Owners.

Yes	No
30	2
93.8%	6.3%

Table G.39: Q47: Are the Product-Owner's responsibilities clear to you?

Yes	No
11	21
34.4%	65.6%

Table G.40: Q49: Do you have problems with Product-Owners?

...can these problems be solved internally?	6	18.8%
...do you need a moderator to solve these problems?”	6	18.8%

Table G.41: Q50: If there are problems with Product-Owners ...

very good (5)	(4)	(3)	(2)	(1)	very bad (0)	positive	negative
5	16	9	2	0	0	30	2
15.62%	50%	28.12%	6.25%	0%	0%	93.75%	6.25%

Table G.42: Q52: How well does the Product-Owner support the work of the team?

very good (5)	(4)	(3)	(2)	(1)	very bad (0)	positive	negative
1	6	11	9	5	0	18	14
3.12%	18.75%	34.37%	28.12%	15.62%	0%	56.3%	43.8%

Table G.43: Q53: Please rate the quality of planning- and preparation of the user-stories.

Yes	No
10	22
31.25%	68.75%

Table G.44: Q54: Are the user-stories introduced early enough?

min. one week before sprint start	in the week before sprint start	directly before the sprint start	in the first sprint planning meeting	during the sprint
2	13	15	2	0
6.3%	40.6%	46.9%	6.3%	0%

Table G.45: Q55: When are the user-stories introduced through the Product-Owner.

very often (5)	(4)	(3)	(2)	(1)	never	negative	positive
5	9	14	3	1	0	28	4
15.63%	28.12%	43.75%	9.37%	3.12%	0%	87.5%	12.5%

Table G.46: Q56: Have you experienced requirement changes during the sprint?

always	mostly	quite often	sometimes	never
1	9	11	11	0
3.1%	28.1%	34.4%	34.4%	0%

Table G.47: Q57: Are the requirements changes reasonable and necessary?

Yes	No
10	22
31.25%	68.75%

Table G.48: Q58: Have you experienced direct access to team-members through the Product-Owner so that the daily tasks are delayed?

very often (5)	(4)	(3)	(2)	(1)	never (0)
1	1	2	4	2	0
10%	10%	20%	40%	20%	0%

Table G.49: Q59: Please rate the frequency of direct access to team-members through the Product-Owner. (The Percentage is related to the number of people who experienced direct access, see Table G.48)

Yes	No
6	26
18.75%	81.25%

Table G.50: Q60: Have you experienced direct access to team-members through a sprint-external Product-Owner so that the daily tasks are delayed?

very often (5)	(4)	(3)	(2)	(1)	never (0)
0	0	1	2	3	0
0%	0%	16.66%	33.33%	50%	0%

Table G.51: Q61: Please rate the frequency of direct access to team-members through sprint-external Product-Owners. (The Percentage is related to the number of people who experienced direct access of a sprint external Product-Owner, see Table G.50)

G.5 Meetings

Yes	No
29	3
90.62%	9.37%

Table G.52: Q63: Are you familiar with the purpose of all meetings?

very happy (5)	(4)	(3)	(2)	(1)	very dissat-isfied (0)	positive	negative
0	6	21	5	0	0	27	5
0%	18.75%	65.62%	15.62%	0%	0%	84.37%	15.62%

Table G.53: Q65: Please rate your satisfaction with the realization of the meetings.

Yes	No
24	8
75%	25%

Table G.54: Q66: In your opinion, are there unnecessary meetings?

Yes	No
18	14
56.25%	43.75%

Table G.55: Q67: In your opinion, is the intended purpose of all meetings met?

far too many	too many	adequate	too few	far too few
1	23	8	0	0
3.12%	71.87%	25%	0%	0%

Table G.56: Q69: Please rate the number of meetings.

very efficient (5)	(4)	(3)	(2)	(1)	very inefficient (0)	positive	negative
0	5	20	7	0	0	25	7
0%	15.62%	62.5%	21.87%	0%	0%	78.12%	21.87%

Table G.57: Q71: Please rate the efficiency of the meetings.

very good (5)	(4)	(3)	(2)	(1)	very bad	positive	negative
0	7	17	6	1	1	24	8
0	21.87%	53.12%	18.75%	3.12%	3.12%	75%	25%

Table G.58: Q72: Please rate how well the time is invested in the meetings.

absolutely positive (5)	(4)	(3)	(2)	(1)	absolutely negative (0)	positive	negative
2	4	8	7	8	3	14	18
6.25%	12.5%	25%	21.87%	25%	9.37%	43.75%	56.25%

Table G.59: Q73: Please rate your attitude towards a ”public demonstration”.

G.6 Project Handling

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
1	11	12	7	1	0	24	8
3.12%	34.37%	37.5%	21.87%	3.12%	0%	75%	25%

Table G.60: Q75: Please rate your satisfaction with the current project handling.

Yes	No
26	6
81.25%	18.75%

Table G.61: Q76: Would you like to know how the projects originate and evolve from their idea to their realization?

very good (5)	(4)	(3)	(2)	(1)	very bad (0)	positive	negative
2	3	10	8	7	2	15	17
6.25%	9.37%	31.25%	25%	21.87%	6.25%	46.9%	53.1%

Table G.62: Q77: Please rate the match between the project handling and the current practice of Scrum.

more than enough	rather more than enough	adequate	rather too few	too few
0	1	13	18	0
0%	3.12%	40.62%	56.25%	0%

Table G.63: Q78: Please rate whether there is enough time for project planning and realization?

very often (5)	(4)	(3)	(2)	(1)	never (0)	negative	positive
6	9	14	2	1	0	29	3
18.75%	28.12%	43.75%	6.25%	3.12%	0%	90.62%	9.32%

Table G.64: Q79: How often did you experience ”forced commitments” in the planning of the sprints.

too long	adequate	does not matter	too short
0	3	4	25
0%	9.37%	12.5%	78.12%

Table G.65: Q80: Please rate the sprint length.

Yes	No
29	3
90.62%	9.37%

Table G.66: Q81: Do you work with full energy on reaching the sprint goal especially at the end of the sprint.

max. effort (5)	(4)	(3)	(2)	(1)	min. effort	positive	negative
8	19	5	0	0	0	32	0
25%	59.37%	15.62%	0%	0%	0%	100%	0%

Table G.67: Q83: Please rate your team's effort to keep the sprint goal.

too much	adequate	too few
0	13	19
0%	40.62%	59.37%

Table G.68: Q84: Please rate the amount of architectural considerations for the projects.

too late	timely	too early
19	13	0
59.37%	40.62%	0%

Table G.69: Q85: Please rate the time of the architectural considerations for the projects.

Yes	No
16	16
50%	50%

Table G.70: Q86: Do the architectural considerations in the technical meetings have an influence on the estimation of the user-stories?

increased	different	decreased
11	5	0
68.8%	31.3%	0%

Table G.71: Q87: Please rate how the estimations are influenced and changed

very often (5)	(4)	(3)	(2)	(1)	never (0)	negative	positive
1	2	10	6	5	8	13	19
3.12%	6.25%	31.25%	18.75%	15.62%	8	40.62%	59.37%

Table G.72: Q88: Please rate the frequency where an architectural veto delayed the review and acceptance of a user-story.

G.7 Defect Handling

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
0	12	14	2	2	2	26	6
0%	37.5%	43.75%	6.25%	6.25%	6.25%	81.25%	18.8%

Table G.73: Q90: Please rate your satisfaction with the current defect-ticket handling process.

absolutely (5)	(4)	(3)	(2)	(1)	absolutely not (0)	positive	negative
0	8	10	8	3	3	18	14
0%	25%	31.25%	25%	9.37%	9.37%	56.25%	43.75%

Table G.74: Q91: Please rate the match of the defect handling process with the current Scrum practice.

Yes	No
12	20
37.5%	62.5%

Table G.75: Q92: Would you prefer a different way of defect handling?

Yes	No
27	5
84.37%	15.62%

Table G.76: Q94: Is there a fair distribution of the defect fixing tasks amongst the teams?

exceeding	adequate	marginal
7	25	0
21.87%	78.12%	0%

Table G.77: Q96:Please rate the number of defect fixing tasks assigned to your team.

Yes	No
7	0
100%	0%

Table G.78: Q97: Does the amount of defect fixing tasks risk the sprint goal?

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied	positive	negative
3	11	12	5	1	0	26	6
9.37%	34.37%	37.5%	15.62%	3.12%	0%	81.25%	18.75%

Table G.79: Q98: Please rate your satisfaction with the defect pre-analysis through the maintenance department.

often	frequently	sometimes	never
0	9	16	7
0%	28.12%	50%	21.87%

Table G.80: Q99: Please rate the frequency of missing the sprint goal due to assigned defect fixing tasks.

Yes	No
26	6
81.25%	18.75%

Table G.81: Q101: Are the defect fixing tasks assigned to the right teams?

very good (5)	(4)	(3)	(2)	(1)	very bad (0)	positive	negative
4	26	1	1	0	0	31	1
12.5%	81.25%	3.12%	3.12%	0%	0%	96.87%	3.12%

Table G.82: Q104: Please rate how well the inter-team support works.

Yes	No
6	26
18.75%	81.25%

Table G.83: Q106: Are there clearly defined guidelines for inter-team support?

Yes	No
6	20
23.1%	76.9%

Table G.84: Q107: If there are no clearly defined guidelines, would it make sense to have them? (The Percentage is related to the number of people who experienced no guidelines, see Table G.83)

Yes	No
1	31
3.1%	96.9%

Table G.85: Q108: Have you already experienced problems supporting another team?

Yes	No
10	22
31.25%	68.75%

Table G.86: Q110: Have the team's sprint goal been risked through support of another team?

often	frequently	sometimes	never
0	2	7	1
0%	6.3%	21.9%	3.1%

Table G.87: Q111: Please rate the frequency of risking the sprint goal through inter-team support.

G.8 Delivery

very happy (5)	(4)	(3)	(2)	(1)	very dissatisfied (0)	positive	negative
4	10	12	2	3	1	26	6
12.5%	31.25%	37.5%	6.25%	9.37%	3.12%	81.3%	18.8%

Table G.88: Q113: Please rate the satisfaction of the co-operation with the delivery team.

Yes	No
25	7
78.12%	21.87%

Table G.89: Q114: Are the Delivery Team's responsibilities clear to you?

Yes	No
24	8
75%	25%

Table G.90: Q116: Are you familiar with the hand-over process of the projects to the delivery team?

Yes	No
8	0
100%	0%

Table G.91: Q117: If not, would it make sens to clearly define the hand-over process to the delivery team? (The Percentage is related to the number of people who have not been familiar with the hand over process, see Table G.90)

Yes	No
17	15
53.12%	46.87%

Table G.92: Q118: Have you experienced risking the team's the sprint goal through support of the delivery team?

often	frequently	sometimes	never
1	8	8	0
3.12%	25%	25%	0%

Table G.93: Q119: Please rate the frequency of risking the team's sprint goal when supporting the delivery team.

absolutely positive (5)	(4)	(3)	(2)	(1)	absolutely negative (0)	positive	negative
4	7	2	6	8	5	13	19
12.5%	21.87%	6.25%	18.75%	25%	15.62%	40.62%	59.37%

Table G.94: Q120: Please rate your attitude towards the idea that the team conducts the acceptance tests at the customer's site?