Doctoral Thesis

Adoption of Agile Software Development Methods in Practice

Ing. Dipl.-Ing. Martin Lechner

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, June 2010

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Deutsche Kurzfassung

Diese Dissertation untersucht die Einführung von agilen Methoden in der Praxis. Hierbei wurde spezieller Wert auf Team bezogene Punkte sowie auf die Einbindung von Usability Werkzeugen gelegt. Die Forschungsarbeiten wurden im Kontext eines akademischen Projektes mit Partnern aus der Wirtschaft durchgeführt. Das Ziel des Projektes war die Entwicklung einer kommerziellen Lösung für den Konsum von Video Clips am Mobiltelefon. Zu diesem Zweck wurde ein Extreme Programming (XP) Team gebildet und im Verlauf des Projektes untersucht.

Die Resultate zeigen einerseits, dass die Einbindung von Usability Werkzeugen in einen agilen Prozess gut funktioniert, wobei diese Werkzeuge an die Bedürfnisse von agilen Prozessen angepasst werden müssen. Andererseits zeigte sich das die Einführung des XP Prozesses problematisch war. Die Daten zeigten, dass gewisse Entwicklungspraktiken von XP schwierig einzuführen waren. Diese Probleme wurden mittels Retrospektive, Literaturrecherche und Mailinglist Analyse untersucht. Die Resultate führten zur Definition eines neuen usabilitybewussten Entwicklungsprozesses. Der Prozess wird beschrieben, es werden Werkzeuge für dessen Einführung bereitgestellt, mögliche Probleme beleuchtet und Lösungen vorgeschlagen.

Abstract

This thesis examines the adoption of agile methods in practice. Special emphasis was put on team specific issues and on the inclusion of usability instruments into an agile process. Research was conducted in the context of an academic project with business partners. The goal of the project was to develop a commercial video streaming application for mobile phones. Therefore, an Extreme Programming (XP) team was created and examined during the course of a project.

The results show that on the one hand, the inclusion of usability instruments worked well but the instruments had to be tailored to the specific demands of agile methodologies. On the other hand, the adoption of the XP methodology proved to be problematic. The data indicates that certain practices of this methodology were difficult to implement. These problems were investigated through reflection, literature research, and mailing list analysis. From the results a new usability aware development process is derived. The process is described, tools for its introduction are provided, possible problems are discussed, and solutions are suggested.

Acknowledgement

I want to thank my supervisor Professor Dr. Wolfgang Slany for giving me the possibility to work in this project and for his patience and help during the writing of my thesis[1]. I also want to thank Professor Dr. Thomas Grechenig who agreed to act as my assessor on a short notice.

I want to thank my teammates Zahid, Harald, Sara, Martin, and Thomas for their valuable input and for sometimes giving me a hard time and something to think about. Special thanks go to Franziska Matzer for her help regarding psychological literature as well as to Christian Schindler for his friendship and his willingness to endure our long discussions about agile methodologies.

I want to thank my teacher Oswald Elleberger for the long years of practice and the introduction to another view on the world which gave me lots of new ideas.

I want to thank my sister and my grandparents for their support during the past years and especially during the past months. Finally, I want to thank all my friends, especially Nicole Klausner who always helped me during critical times and Alexandra Melmer who kept me going during the last days of the thesis.

[1]The research herein is partially conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

This thesis is dedicated to:
My mother Ingrid Lechner who died during my studies,
My grandfather Friedrich Harter who died during the period of this project,
My grandmother Veronika Harter who always supported me despite of these losses.

Graz, 29.06.2010 Martin Lechner

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Software entities are more complex for their size than perhaps any other human construct.
Frederick P. Brooks [Bro95]

Software development is a rapidly growing industry. According to a study from DataMonitor, the industry grew in 2008 by 6.5% and reached a value of 303.8 billion US$ The forecast for 2013 predicts an increase of 50.5% [Dat].

Company	Country	Software revenues (Million US$)	Growth rates
Google	USA	333	455%
Kaspersky	Russia	360	177%
Nintendo	Japan	7245	113%
Omniture	USA	267	102%
Activision Blizzard	USA	4622	73%
Attachmate	USA	313	57%
Emblaze	Israel	296	54%
Take2 Interactive	USA	1452	49%
Autonomy	UK	335	46%
Salesforce.com	USA	959	45%

Table 1.1: The top ten of the fastest growing companies in 2009.

The top ten fastest growing companies in 2009 [wwwj] (see Table 1.1) have an average growth rate of 117% and an average revenue of 1618 million US$. The US Software (SW) industry in 2007 had an annual growth rate of 14% while all US industries together had only 2% growth. This continued the trend that since 2003 this sector has outperformed the rest of the economy. Worldwide in 2008 about 3.6 billion units of software were deployed which was a 20% increase over 2007. The software sales in the "BRIC" markets (Brazil, Russia, India, and China) are projected to rise by 44% to about 22 billion US$ between 2009 and 2012 [All08].

This is not only true for the industry as a whole, but also for individual software products. Software systems are getting bigger and more complex every year with an astonishing speed. The amount of software which is produced and has to be maintained is increasing exponentially. Table 1.2 shows the growth of different operation systems over time in Source Lines of Code (SLOC). Windows alone grew in 12 years to over 13 times its size. Debian did the same in 11 years. The average

Windows			
Year	Operating System	SLOC (Million)	Growth
1990	Windows 3.1	3	
1995	Windows NT	4	1.33
1997	Windows 95	15	3.75
1998	Windows NT 4.0	16	1.07
1999	Windows 98	18	1.13
2000	Windows NT 5.0	20	1.11
2001	Windows 2000	35	1.75
2002	Windows XP	40	1.14
Linux			
Year	Operating System	SLOC (Million)	Growth
1998	Debian 2.0 (Hamm)	25	
1999	Debian 2.1 (Slink)	37	1.47
2000	Debian 2.2 (Potato)	59	1.59
2002	Debian 3.0 (Woody)	105	1.77
2005	Debian 3.1 (Sarge)	216	2.06
2007	Debian 4.0 (Etch)	283	1.31
2009	Debian 5.0 (Lenny)	324	1.14
2000	Red Hat Linux 6.2	17	
2001	Red Hat Linux 7.1	30	1.76

Table 1.2: Source lines of code for different systems [McG03, Whe02, GBRM+09, Sof].

growth of these systems is 1.5 to 1.6 per year, meaning they double their code every 16 months. And they are no exception as a study shows that open source code doubles every 14 months [DR08].

Although the technological challenges are growing, the methods by which software is produced have not changed very much from the beginning of the computer era. At the same time the pressure on the software market has increased. Shorter time-to-market as well as the constantly changing needs of the markets and customers imposes new demands on software development processes. The results of this are more errors, schedule slips, and project failures.

The Standish Chaos Report results shown in Figure 1.1 shows that only a third of all projects is successful[1].

[1]Definition of the terms used in the Chaos Reports:
Successful: On-time, on-budget, and with all features and functions as defined in the initial scope.
Challenged: Late, over budget, and/or with less features and functions than defined in the initial scope.

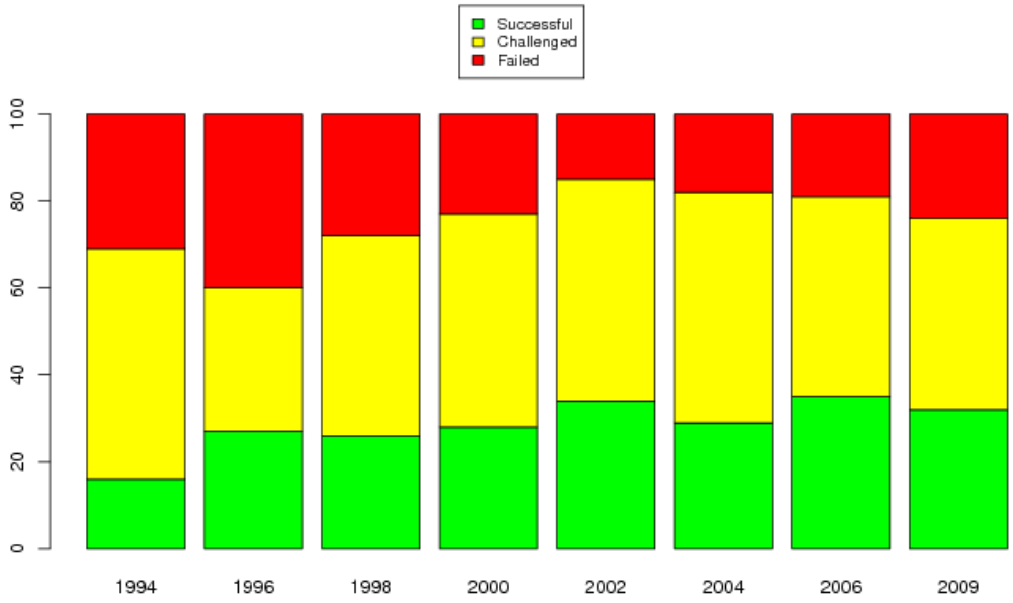


Figure 1.1: Percentages of project success, problems, and failures according to the Standish Chaos Report for IT projects [Sma].

This increasing complexity needs adequate methods for proper project management and software development. Agile lightweight methods like Extreme Programming (XP) or Scrum are designed to cope with the new conditions.

1.1 Challenges in Software Development

There are a number of challenges a SW system is exposed to today and there are many factors which make development complicated. These factors include technical issues, communication problems, environmental issues, ect.

Some of these challenges are system inherent while others are generated by the development method.

1.1.1 The Project Challenge

Most software development is done on project basis. A project is defined as

- "...a unique set of activities that are meant to produce a defined outcome, with a specific start and finish date, and a specific allocation of resources." [Har97].
- "...a temporary endeavor undertaken to create a unique product, service, or result." [PMI04].

Failed: Cancelled prior to completion or delivered but never used.

- "...a unique endeavor and contains risk." [Cha03b]

There are many other definitions, but what most have in common is the fact that a project has to be a unique endeavor. This means that it was never done before and there are many uncertainties and unforeseen events to come. So the definition of "project" itself tells us already that a project is uncertain and not predictable in detail.

1.1.2 The Usability Challenge

As computers and mobile phones have become an essential part of everyone's life, the target audience of software applications has shifted from technical experts to the standard user segment. Accordingly, the typical usage context of Information and Communication Technologies (ICT) has moved from the office to the home. Therefore, usability has become an increasingly important aspect of the adoption and the success of a software product. Today's users expect powerful but nevertheless, easy to use applications. Hence, user-centered design User Centered Design (UCD) processes are needed.

The success of a software development project is associated not only with tools and technologies, but it also depends on how much the development process helps to be user-centered and developer-oriented [HLM+08c]. As intended user interactions strongly influence the internal structure and functionality of the system, User Interface (UI) development cannot be separated from the development of the underlying application [CL99].

For the development of a successful software product it is therefore inevitable to integrate usability into the software development process.

1.1.3 The Flexibility Challenge

On the one hand, Software itself is very flexible. Changes can easily be made. The physical limitations of our environment do not exist in SW products, hugely increasing the possibilities of potential problems or side effects. Therefore, some process is needed to be able to control this flexibility.

On the other hand, the traditional SW processes are often too strict. They are unable to cope with changing environments. Quick time-to-market demands and rapidly changing requirements demand a flexible approach to Software engineering (SWE). Older plan-driven development methods no longer work in today's environment.

1.2 Agile Methods

Many projects fail because of their inability to cope with the changing user requirements. Heavy up-front design without continuous feedback from the customer is a root cause. To have a higher probability of success, the developers need a software development process which has to be flexible enough to cope with the constantly changing requirements and which is also people-oriented.

Agile software development methodologies have emerged in response to these needs. They give more value to individuals, working software and change [Man01]. They have become popular over the years as they are highly adaptable to new situations which make them a promising new way of developing software.

Advantages of Agile Methodologies are:

Lightweight. This reduces unnecessary overhead and frees resources for the tasks at hand.

People centered. People and their communication are seen as the main success factor of the project. This is reflected in practices like the on-site customer and the 40 hour week.

Allow more control over the project. In agile software development, there are two to three levels of planning involved: project scope, release scope, iteration scope. Planning on each scope is not only done once, but incrementally every release and every iteration on a predictable time-frame. This iterative approach of constant planning and re-planning allows the plan to stay in touch with reality. It also allows adjustng to customer wishes or unforeseen events.

Promise to please the customer and deliver in time. This is achieved through integration of the customer into the project and by managing scope instead of the usual requirements documents. The project is executed in a way that after each iteration the system is in a usable and deployable state. This allows the customer to use it immediately and to give feedback as well as suggest improvements. The approach also prevents the risk of having paid months of development time and to finally realize that the project has to be cancelled.

1.3 Criticism on Agile Methodologies

As agile methodologies are relatively new (starting in mid 1990's) they have a hard stand against other well established methods. In recent years, many development teams have adopted XP to evaluate the methodology and to get hands-on experience of an agile development methodology [Tes03]. However, there are also some unsuccessful adoptions and some critical reports.

The term "agile" is often perceived as an excuse for lack of planning and chaotic projects. Agile methodologies can and should be customized to the needs of the project and the project team, but there is the danger to use this as an excuse to get rid of unwanted processes without really adapting the methodology. As this may be true in practice, real agile SW development requires a high level of discipline and also great amount of planning.

Among the less satisfied in the developer community some conclude that it is not the process which lacks in providing a proper base for software development. They think it is a shortcoming on the side of the developers who fail in applying the necessary practices to their full extent and ignore some practices altogether [Rai07].

1.4 Our Research Project

The goal of the project is to increase the quality of software development processes. The reason for large scale research into software engineering techniques has been a formulation of an ideal methodology that can consistently and predictably lead to software development success [MB03]. A recent survey shows that agile software development has seen far better success rates as compared to other methodologies [amb].

Agile methodologies are rather new and therefore being the topic of ongoing research. They are of interest for many academic, research and development organizations as they seem to be more successful than traditional methods. Many experience reports have been presented on agile research with the help of teams using these methods [Tes03] but there is still room for more explorations in the area of agile development methodologies.

Being an emerging agile methodology, XP offers a number of practices, values and principles which are advised to be adopted in order to run a software development project [BA04]. XP is being experimented with in different ways to make it fit to the specific needs of the projects as well as the development teams [Tes03].

Many case studies have been presented by the research community but there is still a need for more experience reports of teams already using XP and also from those who are currently adopting the XP methodology. The data is needed to measure the agility level of software development teams and to determine the success factors of this methodology.

1.4.1 Our Approach

The current research on agile methodologies is mostly done on a case study basis or by examination of properties of one particular practice. Both types of research are often done with students and not with experienced real world developers because the latter are rather difficult to recruit.

Our team has employed the XP methodology to develop a mobile multimedia application. The project under study was a multimedia streaming application for mobile phones that allows making content-based searches for audio and video content in large databases and to play the results on a mobile phone virtually anywhere, at any time. In this project we researched the XP methodology in a context as close as possible to real world conditions. The goal was to use XP to cope with the uncertainties of the project and to research and tailor this methodology for the use in small and medium sized companies.

We propose an iterative and usage-centered approach to UI design and system development in order to cope with the stated problems. Our approach to application development focuses on the adoption of XP and UCD emphasizing iterative UI development involving Usability Engineers (UEs) and end-users.

Project goals

The business goal of the project was to develop a multimedia streaming application for mobile devices with an emphasis to utilize huge archives of TV programs, radio programs, and other documentary and entertainment content.

The scientific goal was to research XP by developing in an XP way, meaning that the developers were at the same time researchers. Also the aspect of usability in XP and more generally in agile projects was examined. For this reason we worked together with a usability center.

Another research goal of our project was to apply usability test procedures for mass-market applications on mobile phones. At present, usability testing for mobile phones is cumbersome and too expensive for small and medium sized enterprises. An objective of this project was to automate certain parts of the usability testing procedures and to provide a testbed for effective and efficient mobile usability testing. Special emphasis was placed on the adoption of agile software development methodologies, in particular XP, for mobile phones and their user-interfaces.

XP Motivation

As development methodology XP was selected with an intention to use it in a progressive manner: conscious of applying each practice, that can be applied, and looking for the improvement and optimization of the whole XP process. This was the basis for a profound academic research and for using the flexibility of XP for developing a commercial successful application.

The Project Setting

The project team consisted of six PhD students, five developers (a mix professional programmers and members from academia) and one business person. We had cooperation's with partners from industry and potential customers. The business person together with the project supervisor acted as sales men, project coordinators, and on-site customer.

Evaluating the Process

Each developer, also being a researcher, took part in the analysis, development, and improvement of XP. The data required for analyzing the process performance was collected by actually implementing each practice and record the results. Different tools for planning and for empirical data collection have been applied during the project to analyze the performance of the used XP practices. Iteration and release velocities were recorded to visualize the throughput performance of the team over a period of time, reflection notes were taken, subjective questionnaires were used, and code based metrics were applied.

1.5 My Thesis: Adopting Agile Methods

Being agile, it is inviting to mold and reshape the methodology according to the requirements of the project. However, useful changes can only be made if the basic concepts are internalized and understood. Our experience showed that even though the XP methodology is simple, a serious effort had to be made to apply its practices. It was difficult to maintain a balance between agility and discipline in order to get the maximum benefits from the method.

In this thesis I use the data we derived and the experiences we made during the duration of our project. Agile methods in general are covered with special emphasis on XP. The new aspect of usability in agile projects is discussed in detail. It is explained how to fit this two methodologies together and the combination with various usability instruments is investigated. This provides insights how to introduce an agile and usability aware processes in companies to improve their project success rates. Furthermore, to go beyond a mere case study with personal experiences, a new approach of data-mining an XP mailing list repository is evaluated.

During the project lifetime we encountered many problems and pitfalls which are likely to occur also elsewhere. I give explanations for those problems and offer solutions how to avoid them. Furthermore, I outline a usability aware agile process and provide some suggestions and tools for its introduction.

1.6 List of Publications and Collaborations

Within the context of this thesis, the following scientific papers were published:

2007: User Interface Design for a Content-aware Mobile Multimedia Application: An Iterative Approach [HLM+07b]

2007: The Sustainable Application of Leadership and Teaming in HRM - Models for the Today's Organizations' Needs – a Meta-Level Perspective [HLM+07a]

2008: User Interface Design for a Mobile Multimedia Application: An Iterative Approach [HLM+08b]

2008: Optimizing Extreme Programming [HLM+08a]

2008: Probing an Agile Usability Process [WTS+08]

2008: Integrating Extreme Programming and User-Centered Design [HLM+]

2008: XP Team Psychology - An Inside View [Lec08]

2008: Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application [HLM+08c]

2009: Concept and Design of a Contextual Mobile Multimedia Content Usability Study [HLM+09]

As the topic of our research was XP, we applied the XP practices Whole Team and Pair Programming (PP) also to the research part of our work. Most of our research was thus done jointly by the whole team, and accordingly publications were also written in PP style by all members of the project team. Two persons were sitting in front of one computer and writing a part of the paper while another pair

was working on another part. We also used frequent pair changes and Collective Code Ownership, and consequently each person worked on all parts of the paper. Reflecting this particular setting, it was decided to list the author names of the publications in alphabetical order. Therefore the position of Zahid Hussain's name as the first author is based on this alphabetic ordering and his family name, and not because of any distinguished role of him in the carrying out of the research on which the publications were based, nor on a leading role in the process of writing up the text of the paper itself.

Paper [WTS+08] was initiated by our partner CURE (Center for Usability Research and Engineering). We contributed the XP relevant part to this paper and CURE had the lead role concerning the Usability aspects of this paper, the paper describing our joint efforts in unifying these two aspects of software development. Therefore, the authors in this paper are not sorted alphabetically, also due to the request of this external project partner and the differing author placement culture on which they insisited for this particular publication.

My solo publication [Lec08] reflects my special interests in the project. I am mainly concerned with XP, factors which affect its introduction and use, and the interaction between its practices with each other as well as with other processes like Usability. These aspects are my personal singular contributions but stand on an equal footing compared to the research carried out jointly in our team.

1.7 Paper Structure and Contributions

This introductory chapter provides an overview of the thesis. The background, the motivation for the research, as well as the context is explained.

Chapter 2 gives an overview about software development processes with a focus on agile methods. Parts of this chapter are derived from my master thesis [Lec05].

Chapter 3 provides some background information about usability methods and processes. Chapter 4 describes the structure of the M3 project and the environmental context. Chapter 5 describes the basic concepts of the M3 application. Chapter 6 describes the evolution of the application in the course of the project. Chapter 7 describes the M3 XP process and Chapter 8 deals with the M3 Usability Inclusion methods. The material used in these chapters is mainly from our publications [HLM+07b, HLM+08a, HLM+08b, HLM+08c, HLSS08, Lec08, WTS+08, HLM+09, HLM+] as well as from our project reports.

Chapter 9 is a slightly modified version of publication [Lec08] and deals with the team related problems we encountered. Chapter 10 describes an approach to extract XP context factors with a data-mining approach. Both chapters try to identify problems which occur in an agile process as well as their reasons and environmental influences. In Chapter 11 I propose a process derived from our experiences and explain how this process can be introduced in a company. Special emphasis is put on possible problems. Tools are provided to cope with those issues during the introduction.

Finally, Chapter 12 provides a summary about the results of this thesis. Furthermore, an outlook about future work is given.

Chapter 2

Software Development Processes

If terminology is not corrected, then what is said cannot be followed.
If what is said cannot be followed, then work cannot be accomplished.
If work cannot be accomplished, then ritual and music cannot be developed.
If ritual and music cannot be developed, then criminal punishments will not be appropriate.
If criminal punishments are not appropriate, the people cannot make a move.

Therefore, the Superior Man needs to have his terminology applicable to real language,
and his speech must accord with his actions.
The speech of the Superior Man cannot be indefinite.
Analects of Confucius, Book 13, Verse 3 (James R. Ware, translated in 1980)

Software development processes are an attempt to structure and standardize development to make the outcome of a project plan-able and predictable. There are many process models available and being used ranging from heavyweight to agile. In accordance with the project circumstances the actual development process has to be tailored to the specific needs.

The choice of the right process depends on various factors: Risk level, requirements stability, time-to-market, etc. The higher the involved risk, the more structured and heavyweight the process has to be to ensure safety. The more unstable the requirements are the more iterative and agile the underlying process has to be to cope with the changes. The shorter the time-to-market the more lightweight and iterative the process should be to avoid administrative overhead and provide deployable software after each iteration [Lec05].

In this chapter a short overview about Software (SW) Development methods is given. Because of the huge variety of existing methods only the most common and those which are referred to in the context of this thesis are mentioned.

2.1 The Waterfall Model

The waterfall model was the first structured approach to system development. It gained in popularity during the 70s and allowed for a certain amount of order in the process. It is based on hardware engineering models and is widely used in defense and aerospace industries. Although the model in general is the same, there are slight variations of the phases included. The waterfall model is just a time-ordered list of activities to be performed to obtain a software system. Therefore, phases can be split, joined, or new ones can be inserted as necessary to match the process.

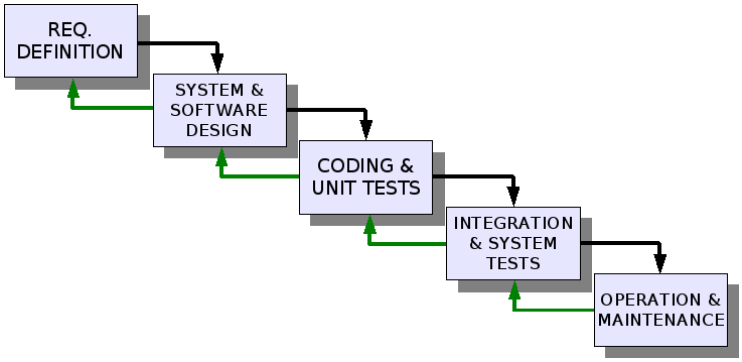


Figure 2.1: Waterfall model [Sch10].

The most important characteristic of the waterfall model is that there are only transitions between two adjacent phases. If one phase is completed, the next is started. If problems occur it is possible to go back one step to solve them. Its rigidity is one of the major disadvantages of the waterfall model as during development it is often necessary to go back more than one step. Although this is possible in this model it is very undesirably. Therefore, a number of criticisms have been brought up against this model:

- Problems are not discovered until system testing.
- Requirements must be fixed before the system is designed – requirements evolution makes the development method unstable.
- Design and code work often turn up inconsistencies in the requirements, missing system components, and unexpected development needs.
- System performance cannot be tested until the system is almost finished. Lack of capacities may be difficult to correct.

This model was and remains relatively popular because of its pretended predictability. However, because of its limitations it is quite rare that a software project will exactly follow the steps of a waterfall model [Lec05].

2.2 The V-Model

The General V-model is a logical product model and is neither a physical product model nor a life cycle model. It describes the contents of the products which have to be created during a software project and their relationships on a very high level. In a real project the physical structure of these products will often be quite different.

However, due to its very general description, products used in a life cycle model can always be matched with the products contained in the V-model (as seen in Figure 2.2). It is also possible to change the number of phases to get a better match to the life cycle. Therefore, it can be seen as a generic product model.

Basically the V-model is a waterfall model. The individual phases of the waterfall are arranged in a fashion that allows highlighting the difference between validation and verification. Another point that is shown clearly by this arrangement is the

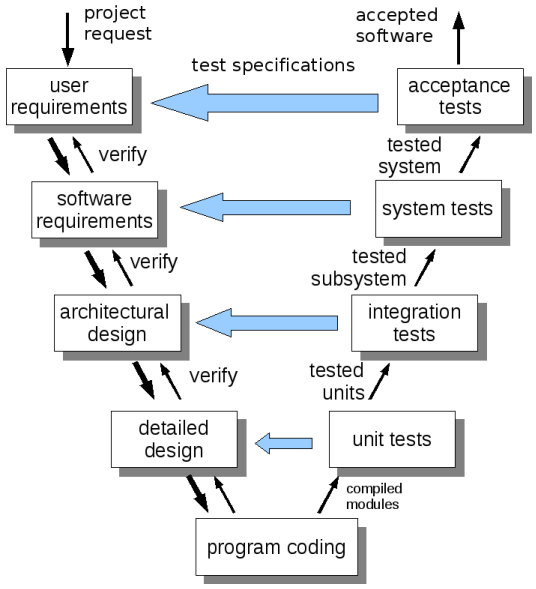


Figure 2.2: V-model for Software development [Sch10].

correlation between the individual design and test steps.

The model can be divided in a left and a right part and includes also analytical activities (verification and validation).

Left: The left side of the V shows the results of the development. This can be seen as the production cycle.

Right: The right side of the V represents the construction of the final system starting with individual components. This can be seen as the testing cycle.

Verification: Verification means to check that the results of a development step are as intended. Each step on the left side includes verification against the previous step to ensure that the implementation is correct. The question asked is: "*Are we building the product right?*"

Validation: Validation is to determine whether an implemented system fulfills its requirements. At each level on the right side there is a validation between the specification from the left side and the actual implementation. The top level validation is the system validation against the user requirements. The question asked is: "*Are we building the right product?*"

The V-model presents a nice view of a split-and-merge philosophy. On the left side, the split is done from user requirements down to individual units. On the right side, these modules are combined progressively into a software system.

A benefit of the V-model is that it shows clearly how processes on the left and the right side of the V can be done partly in parallel. As soon as a step on the left side is completed, the preparations for the associated step on the right side can begin. For example when the detailed design phase is finished, the unit tests can be written. Therefore, the appropriate test plan or test procedure is ready by the time when it is needed. This can save considerable time and money.

A more general view to the V shape also tells that both parts have more or less the

same number of phases. This indicates that testing may require much more time and resources than normally allocated. An effort of 20-30% of the total development man-days is a good estimate [Gre05], a fact that is rarely considered, but has shown to be true in many cases [Lec05].

2.3 Iterative Development

I find that teams can grow much more complex entities in four months than they can build.
Frederick P. Brooks [FPB87]

Although at present iterative development is often thought to be a new invention it has a long history. One of the first and most influential proposals on iterative development was the "plan-do-study-act" (PDSA) made in 1930 by Walter Shewhart, a quality expert at Bell Labs. Later in the 1940s W. Edwards Deming began vigorously promoting PDSA. Also for the X-15 hypersonic jet project in the 1950s iterative development was used and the practice was considered a major contribution to the X-15's success [LB03]. Iterative development can be used in many different ways. Some established iterative development models are:

Spiral Design: Go through waterfalls, starting with a very rough notion of the system and becoming more detailed over time. Incorporates risk assessment in each iteration.

Modified Waterfalls: Waterfalls with overlapping phases, waterfall with sub projects, etc.

Evolutionary Prototyping: Start with initial concept, design and implement an initial prototype, iterate as needed through prototype refinement until acceptable, complete and release the acceptable prototype.

Staged Delivery: Go through concept, requirements analysis and architectural design, then implement the pieces, showing them to the customer as the components are completed – and go back to the previous steps if needed.

Evolutionary Delivery: A cross between evolutionary prototyping and staged delivery.

2.4 Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions	*over*	*processes and tools*
Working software	*over*	*comprehensive documentation*
Customer collaboration	*over*	*contract negotiation*
Responding to change	*over*	*following a plan*

That is, while there is value in the items on the right,
we value the items on the left more.

Kent Beck	*James Grenning*	*Robert C. Martin*
Mike Beedle	*Jim Highsmith*	*Steve Mellor*
Arie van Bennekum	*Andrew Hunt*	*Ken Schwaber*
Alistair Cockburn	*Ron Jeffries*	*Jeff Sutherland*
Ward Cunningham	*Jon Kern*	*Dave Thomas*
Martin Fowler	*Brian Marick*	

Manifesto for Agile Software Development [Man01]

The next logical step was the invention of the agile software development methods. In software engineering, agile software development or agile methods refer to low-overhead methodologies that accept that software is difficult to control. They minimize risk by ensuring that software engineers focus on smaller units of work. One way in which agile software development is generally distinguished from "heavier", more process-centric methodologies, for example the waterfall model, is by its emphasis on values and principles, rather than on processes. Typical development cycles are one week to one month. At the end of each cycle the project priorities are re-evaluated. This is a feature that is shared with iterative development methodologies and most modern theories of project management. Additionally, agile methods usually have implicit or explicit mechanism to adapt the process itself.

Agile methodologies are becoming more and more popular as they define in themselves the whole social structure which is needed to run a development process in a productive way. They are designed to boost up the developers' efficiency by giving them enough time and space to catch-up with changing user requirements as well as with the evolving technology, paying also attention to the social factors during the whole development process. Cockburn [Coc] considers the people and their characteristics as the dominant, first-order project success driver.

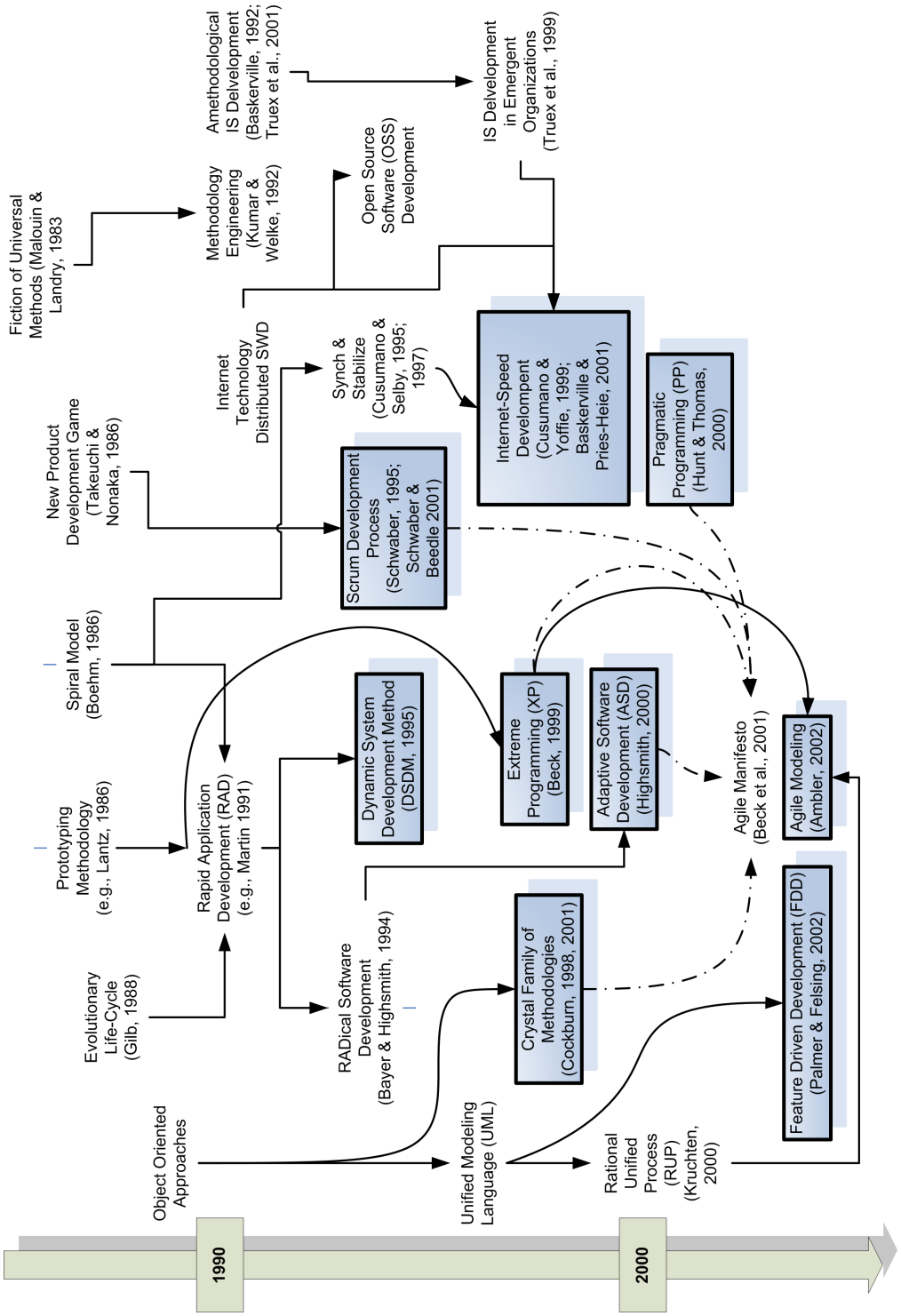


Figure 2.3: The evolution of agile methods according to [AWSR03, Sch10].

2.4.1 Scrum

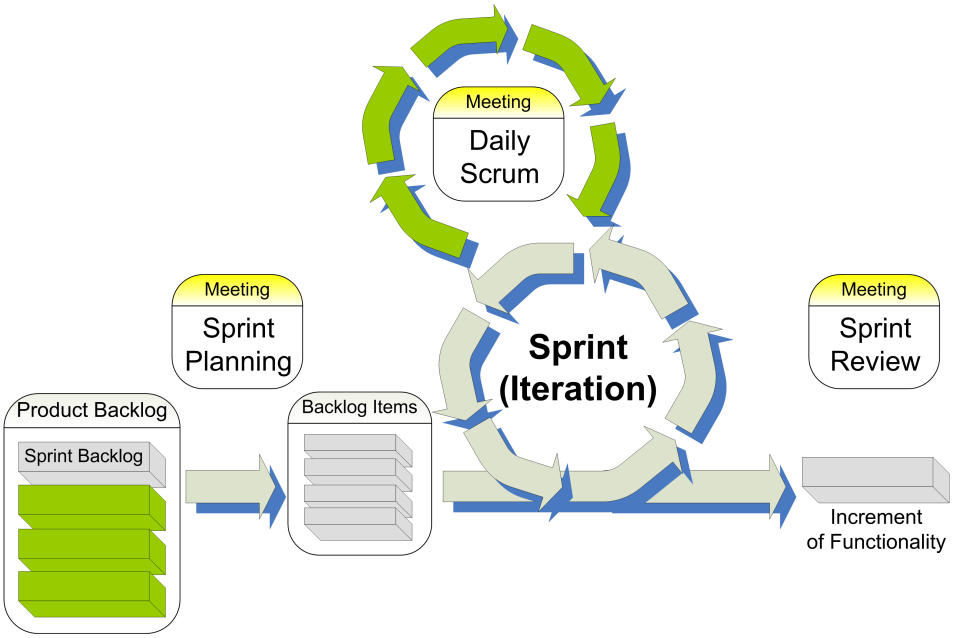


Figure 2.4: A Scrum process overview [Sch10].

Scrum is a method to manage projects. It is not limited to SW development and therefore, makes no concrete assumptions about how the development is actually carried out, but has strict rules how the management should be conducted. Although used earlier [Sut04], Scrum was first introduced as a formal development process by Ken Schwaber and Jeff Sutherland at OOPSLA 1995 [1] [Sch95].

A Scrum team consists of a Product Owner, a Scrum Master, and a team of 7±2 developers. These persons are called "pigs" while everyone else is a "chicken". The Scrum Master is responsible for teaching and enforcing the Scrum rules. She is coaching the others, protects the team from external influences, and removes impediments within the team and the organization.

The Product Owner is the only responsible person for decisions regarding the product. Her job is to prioritize and maintain a Product Backlog and to act as the interface to the team. Therefore, she communicates with the stakeholders, the customer, and the team.

The team of developers is cross-functional and must have all the skills necessary to actually carry out the work at hand. It is self-organizing and there are no specialized subgroups as everyone is expected to help out and learn new skills if required. The team composition may change after a Sprint but this is not advisable as the gain form the self-organization is lost.

The Scrum Master as well as the Product Owner can be team members too, with the exception that the Scrum Master can never be the Product Owner. However, sharing of roles is not recommended as these roles by themselves are challenging

[1] Object-Oriented Programming, Systems, Languages & Applications.

which tends to lead to conflicts with the development work.

Scrum development is done in Sprints which are Iterations of ≤ 30 days in which the development occurs. A Sprint planning meeting is held at the beginning of the Sprint and consists of two parts. In the first part, prioritized and estimated items from the Product Backlog are selected by the Product Owner and presented to the team. The team asks for clarification if necessary and finally commits to a set of items. In the second part, these items are broken down into development tasks which form the Sprint Backlog. Before the Sprint planning meeting, estimation and re-estimation of Product Backlog items have to be done in a separate estimation meeting which occurs usually during the previous Sprint.

During the Sprint, the Scrum Master ensures that the Sprint Goal is not changed [KS10], providing a stable development situation for the Sprint period. The Product Owner can cancel a Sprint, but this is rarely necessary due to the short duration. It should be avoided as it can traumatize a team [KS10].

At the end of each Sprint, a Sprint Review as well as a Sprint Retrospective Meeting is held. The Sprint Review meeting allows to asses the status of the project. During this meeting the team talks with the Product Owner and the stakeholders about the Sprint and demonstrates the completed functionality. It provides the background for the next decisions of the Product Owner and the prioritization of items during the next sprint planning meeting.

The Sprint Retrospective meeting is concerned with the improvement of the development process itself. The last Sprint is analyzed and possible improvements as well as impediments are identified. These items are also prioritized and implemented during the next Sprint. To allow an open communication, only team members are allowed to participate.

The team itself has an additional daily meeting to synchronize called “Daily Scrum” which is only for the team members. Others, like the Product Owner or stakeholders, can attend this meeting but are not permitted to talk. The Daily Scrum is a short, ten to fifteen minutes, meeting of the team members and the Scrum Master. Each team member explains briefly to the others what she has done, what she plans to do, and what obstacles she sees. The role of the Scrum Master is to keep the meeting short and to enforce the Scrum rules.

The Scrum process adapts a project to an unstable environment. This is achieved by the iterative approach which allows changing directions after each Sprint and by adapting the length of the Sprints to get the necessary flexibility. Additionally, the explicit role of the Product Owner as an interface to the team provides a single point of responsibility and helps to avoid confusion. However, the Scrum process only concerns the organizational part of a project. It does make no assumptions whatsoever about the underlying development practices. Therefore, it is often combined with certain Extreme Programming (XP)practices.

2.4.2 eXtreme Programming (XP)

XP is one of the mostly used agile processes in the SW industry and aims to continuously deliver quality software by satisfying the customer. The XP methodology

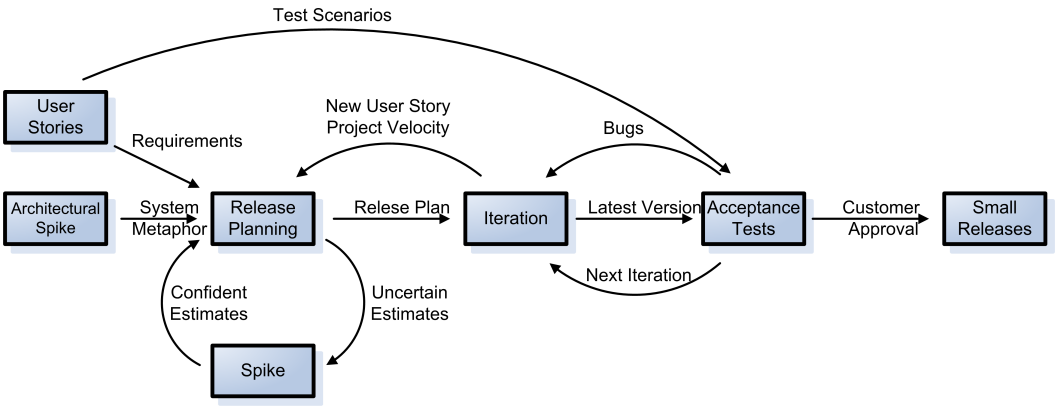


Figure 2.5: An XP process overview [Sch10].

was formulated by Kent Beck, Ward Cunningham and Ron Jeffries. In March 1996 Kent Beck started a project at DaimlerChrysler using new concepts in software development [2]. The result was the XP methodology.

Contrary to Scrum, XP is a development methodology targeted only at SW development. The starting point was to find out what made software easy to create and what made it difficult. Kent Beck came to the conclusion that there are four factors to improve a software project: Communication, Simplicity, Feedback and Courage are the values sought out by XP programmers [Bec99a, Bec99b]. In the second edition of his book, Beck added "Respect" as a fifth value [BA04].

Improve communication: Improper communication is one of the root causes of failures within all kinds of projects. Results are schedule slips, botched requirements, faulty development assumptions, and the like. XP addresses these problems by stressing good communication between all project stakeholders – customers, team members and project managers – on a consistent basis. A representative from the customer should be present on site at all times to answer questions and clarify project requirements; thus allowing a tight cooperation with the customer and immediate feedback. Programmers are expected to work simultaneously in pairs with each programmer reviewing the other's work.

Communication and collaboration between customers, business partners, developers, and other stakeholders enhances the overall team efficiency [McN]. The value of communication is expressed by the XP practices of Pair Programming (PP), metaphor, Informative Workspace, Simple Design, Onsite customer, Planning Game, and Coding standards [HT04]. Other factors in communication are the use of whiteboards, positioning and sharing of desk facilities to facilitate PP, stand-up meetings, developers buying-in to the concepts of the rules and practices of XP, and Collective Code Ownership [GH01].

Seek simplicity: XP tries to keep SW development simple by allowing the programmer to concentrate only at the current task. This is expressed by the following slogans: "Do the Simplest Thing That Could Possibly Work" and "You Aren't

[2] http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,66192,00.html

Going to Need It" (known as YAGNI). The result is a more focused approach. The programmer can implement what she has to, without thinking about future expansions. Even if it seems to be clear now that some code has to be written a few minutes later, it is not done yet if it is not needed for the current task. If necessary, these changes are made later during the implementation of the task which requires them. The necessity of these changes is determined with the "If it stinks, change it" principle. Please note that this is in total opposition to traditional approaches to software engineering!

Get feedback on how well you are doing: A basic idea of XP is that there should always be some running system that delivers information about itself in a reliable manner. Automated tests give immediate feedback about the state of the project and the effects of changes. Feedback on process level is gained at the stand-up meetings and the reflection and planning meetings. Feedback serves as a catalyst for change and an indicator of a project's progress. Code refactoring is derived from this value.

Be able to proceed with courage: XP acknowledges that projects are ultimately people centric. It is the ingenuity of people, and not of any particular process, that causes projects to succeed. Courage also manifests itself in the features of feedback and refactoring, as described in the XP principles.

Respect: The contributions of each person on the team need to be respected, including all the stakeholders.

Being a lightweight agile method, XP has the advantages of: on-time delivery, co-located team, relying on the team members' knowledge rather than documentation, optimized resource investments, short release cycles, working high quality software, tight customer integration, incremental design, constant communication and coordination, rapid feedback, continuous refactoring, PP, and Test-driven Development (TDD) [Bec99b, BA04, AWSR03]. Also, being people oriented it defines the whole social structure which is needed to run a development process in a productive way.

XP Practices

"XP is a collection of well-known software engineering practices. XP aims at enabling successful software development despite vague or constantly changing software requirements. The novelty of XP is based on the way the individual practices are collected and lined up to work with each other" [AWSR03].

Being people-oriented, XP cares about and addresses social factors like teamwork, communication and collaboration, learning-and-sharing, motivation, enthusiasm, and interaction in most of its practices. The XP practices were developed and tested around the humanistic phrase that "Software is built by human beings. In the end keep the human being focused, happy, and motivated and they will deliver" [BF00a, SAHG06].

Planning Game: During the Planning Game the customer together with the developers decides what functionality should be implemented during the next iteration or release. The functionality is described in so called User Stories.

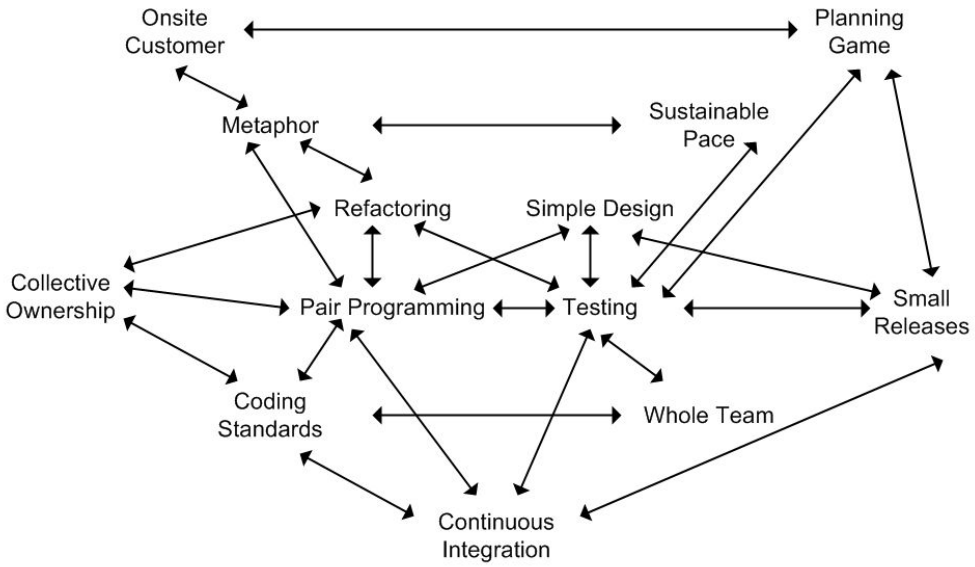


Figure 2.6: Interaction of the XP practices [Bec99b].

During the Planning Game it is the task of the developers to estimate the stories while the customer prioritizes by deciding which stories should be done in which order. The velocity, the sum of the completed-story estimates of the last iteration, together with the current story estimates determines how many stories can be selected.

Testing: This means mainly test-first and has to be understood differently than the traditional testing approach. Acceptance tests are specified by the customer to capture the expected behavior of the system, while unit tests are written by the programmer during implementation. These tests are also the basis for Continuous Integration and Refactoring. As in XP these tests are written before the implementation, they are actually a specification and design technique rather than a pure a testing technique.

Pair Programming: All production code is developed by two programmers sitting side by side in front of one computer. This is a quality improvement process through continuous peer review. Also the skill transfer is facilitated and the truck factor, the possibility of project failure because one key person is lost (e.g., overrun by a truck), is reduced. Therefore, pairing partners have to be switched regularly and everyone has to pair with everyone else. This practice is closely related to Collective Code Ownership and Coding Standards.

On-site Customer: The customer is not seen as an external entity, but is integrated into the development team. This leads to increased communication and mutual understanding. Later this practice was renamed to “Whole Team” because also other roles like tester, QA, etc. can be included into the team, and to underline that the reason for this practice is that the involved people work together as a whole instead of different entities.

Simple Design: The system should be designed in the most minimalist way to satisfy

the current requirements at the current moment without anticipating future development. If necessary changes occur the design is changed or int he worst case completely redone. As this approach sounds shortsighted, it is based on the facts that most anticipated functionality will never be used and that simple system are more changeable than complex ones because they are easier to understand. Even if the simple design has to be redone the effort was not in vain. Through the first design important information about the system was gathered making the consecutive design more suitable. As this approach demands constant changes of the system, this practice is only possible in conjunction with Testing and Refactoring.

Refactoring: Refactoring is necessary to avoid a deterioration of the design as new functionality is constantly added. It means to change the internal structure of a system without changing the external behavior [Fow99]. To be able to do Refactoring, the testing practice is necessary as tests have to be present to verify the external behavior.

Metaphor: "The transference of the relation between one set of objects to another set for the purpose of brief explanation" [web]. The metaphor should provide a common vision about what the system is about and how it could be described to achieve better communication.

Small Releases: This practice should ensure a steady and visible progress of the system. Small releases give the developers a near and concrete target which motivates work. The customer gets improved working versions of his software frequently which builds trust.

Continuous Integration: As many problems in traditional software projects manifest themselves during system integration this is done continuously through an automated process to get feedback as early as possible. Also for this practice the testing practice is a precondition to get the maximum benefits. Through Continuous Integration (CI) the whole system is compiled and tested continuously, usually once a day or after each committed code change. Therefore, the higher the test coverage and the better the quality of these tests the more errors can be detected at an early stage.

Collective Code Ownership: As there is no individual code ownership, every piece of code can be modified by everyone if necessary. This facilitates the distribution of knowledge about the system through the team which reduces the truck factor. Additionally, the shared responsibilities reduce "blame games".

Coding Standards: A common coding standard expresses that the team is indeed "a whole" and is the precondition for Collective Code Ownership (CC) as everyone is able to read the code.

Sustainable Pace (was: 40-hour Week): The progress in an agile process should be steady and continuous. Therefore, the pace must be sustainable which is not possible for extended periods of overtime. Overtime is a demotivating factor, tends to destroy teams and the social lives of the developers, is usually compensated by undertime, and produces more errors [DL99, HT04]. Therefore, overtime should be avoided as much as possible.

Stand-up meeting: The Stand-up meeting is neither in [Bec99b] nor in [BA04] mentioned as a core practice but nevertheless it is an important practice in XP

and other agile methodologies. The stand-up meeting is a short daily meeting to communicate the project status inside the team and to plan the immediate future. In the meeting everyone shortly describes what she has done since the last time, what she intends to do till the next meeting, and if problems occurred. This informs the other team members about the progress. It is to note that discussion and problem solution are not part of this meeting. Discipline is necessary to adhere to these restrictions which are necessary in order to keep the meeting short and productive.

Some XP Terms

User story: A User Story in XP is a small narrative description of a feature from the users' point of view which should fit "on the back of an envelope". Only the most important facts are mentioned as the concrete details are negotiated with the customer during development of the story. The story should be of a size that the team can implement several in one iteration. E.g. if the iteration lasts one week a story should not exceed 1-2 days. If the story is larger it has to be divided into several smaller ones.

Velocity: The velocity determines the possible amount of stories which can be fitted into one iteration or release. It is defined as the sum of the estimates of all finished stories of the last cycle (iteration or release). This measurement is based on the assumption that the team can do the same workload and that the estimation error stays the same. It is to note that for different scopes (e.g., release and iterations) different velocities are used.

Estimates: Stories are estimated by the whole team to get a common understanding. The form of estimates varies from ideal time over story points to gummi bears. Estimates in ideal time are based on time and assume the ideal case without problems or interruptions, while estimates in story points or gummi bears are relative measures compared to earlier stories. While it is generally easier to estimate in ideal time, this estimate can lead to confusion with real time. This might provoke false expectations from management and even inside the team.

Co-location: Also referred to as "sit together" means that the whole team shares the same room. Together with the practice of PP, where the developers have to talk to each other, this practice supports the informal and subconscious flow of information between the team members as everyone can hear what is going on and if potential problems occur. Co-location together with pair programming tells everyone *how* things are done. This augments the practice of stand-up meetings where it is explained *what* is done.

Informative Workspace: Informative workspace means that the workspace should be designed in a way which supports the flow of information about the project. This is achieved through big, visible diagrams, story boards, extensive use of whiteboards, etc. which are constantly kept up-to-date. The goal is that the room itself

becomes part of the information structure. One important factor is that these artifacts exist physically and not in the computer. This practice is closely related and overlapping with the practice of co-location.

Story Board: A story board is a place where the User Stories are displayed. It is part of the Informative Workspace and should be easily visible and accessible. The story board shows the progress of the application because the status of the stories can be seen at a glance. It displays which stories exist, which are scheduled, which are worked on, and which are already finished.

Chapter 3

Usability

”Usability is most often defined as the ease of use and acceptability of a system for a particular class of users carrying out specific tasks in a specific environment.” [Hol05]

The most important factor for the success of a software application is user acceptance. “An inherently usable and technically elegant application cannot be considered a success if it does not satisfy the end-users’ needs. End-users are often left out of the development process” [McN]. A usable software application should focus on its end-users, theirs goals, and their satisfaction, therefore increasing the usability of the application for the people which actually use it.

3.1 Usability on Mobile Devices

Usability of a mobile application is an important ongoing research issue. Numerous studies address Usability Engineering / User Centered Design (UCD) issues for mobile applications [BFB07, HSN07, KS04]. It is also a major concern for our project as mobile phones are the primary target platform.

The inherent interface limitations of mobile phones strongly constrain the choices of User Interface (UI) and interaction design. Special attention has to be paid to the constraints of small screens [KMS05], possibly unfavorable lighting conditions, and limited text input capabilities. Appropriate UIs are therefore crucial for enhancing the user experience and enabling the effective consumption of mobile content [SY06].

3.2 User Centered Design (UCD)

“Usability measures the quality of a user’s experience when interacting with a product or system; UCD is an approach for employing usability” [Usa].

UCD is a design approach focused on the information about the people who are the actual users of the product. This user focus is maintained by considering this information during planning, design and development of a product [W3C04]. Holzinger emphasizes that every software practitioner should be aware of different usability methods and apply them according to specific circumstances of a project [Hol05].

User-interface development cannot be separated from the development of the underlying application. Intended user interactions strongly influence the internal structure and functionality of the system [CL99].

Each of the processes mentioned in Chapter 2 has different implications when it

comes to the inclusion of usability engineering. For the advice on usability inclusion, we focus on agile methods with special emphasis on the lightweight iterative Extreme Programming (XP) approach.

3.3 Usability and Agile Methods

Recently there has been an increasing interest in integrating agile and user experience/UCD methodologies both in the agile community and the Human Computer Interaction (HCI) community. In fact there were special tracks and dedicated workshops in the Agile 2008 conference[1] as well as in the CHI 2008 conference[2].

The integration of agile methods with HCI practices was discussed by Kent Beck and Alan Cooper in 2002, concluding that both interaction design and XP have strengths to be combined [Nel02]. There are several studies examining various aspects of the integration of both methodologies. Patton [Pat02] has described the way of incorporating interaction design in an agile process. Chamberlain et al. [CSM06] have conducted an ethnographic field study to explore a framework for integrating agile methods with UCD. In their case study, McInerney and Maurer [MM05] interviewed three UCD specialists for integrating UCD within agile methods and reported positive feedback. Ferreira et al. [FNB07a] investigated several projects for the relation of UI design and agile methods. Fox et al. [FSM08] also conducted a qualitative study that describes how the agile methods and UCD are integrated in industry. Obendorf and Finck [OF08] report their experience of combining XP and scenario-based usability engineering.

There already exist approaches of integrating agile methodologies and Usability Engineering / UCD [HES+05, FNB07a, MM05, CL03, MRH07]. Memmel et al. point out that "When Usability Engineering becomes part of agile software engineering; this helps to reduce the risk of running into wrong design decisions by asking real end users about their needs and activities" [MRH07].

The focus of both approaches on users makes it possible to integrate them [GGB03a]. This integrated process allows combining the benefits of both methodologies and makes it possible to reduce the shortcomings of each because agile methods need to know their true end-users and UCD benefits from a flexible and adaptive development methodology which runs throughout the project life-cycle [Tox05].

Some experts doubt that agile processes lead to true user-centered design [Hud05]. These statements suggest that agile methods and UCD do not fit together. But this perception is simplistic and misguided which has already been shown in practice where success is reported [MM05]. We can see succeeding practitioners combining usability/UCD and agile methods by varying approaches [Pat02, CSM06, HES+05, MM05, HS06, Sy07, FNB07a, FNB07b, FSM08, Amb08, WTS+08, HLM+08c, HMS+09, MS09, BJK09].

[1] http://www.agile2008.org/
[2] http://www.chi2008.org/

3.4 Similarities between XP and UCD

In our project we us XP as development methodology and combine it with UCD. The core values of XP and UCD [GGB+03b] are applied to solve different issues: In XP a simple implementation, fulfilling the minimum requirements of the application is created and iteratively extended. UCD tries to continuously improve the usability of the user interfaces. However, when comparing some of the core values, it seems obvious that the two development processes can benefit from each other's practices.

3.4.1 On-site Customer

UCD is an approach to user interface design focusing on end-users throughout the planning, design, and development stages of a product [W3C04]. In UCD, "all activities are focused on providing business value through ensuring a useful, usable and engaging product. The customer is not only defined as the project stakeholder, but the end user as well" [McN].

In contrast, the Manifesto for Agile Software Development [Man01] does not clearly demand end-users as customers. Agile development processes involve a customer as a business representative who is responsible to specify the business value of user requirements. He is co-located with the programmers in order to answer domain-specific questions and give feedback on the system. But this customer needs not necessarily to be a real end-user, and rarely end-user takes the customer role [CSM06]. This is a shortage as there is evidence that coordination only with the customer does not ensure good usability [JA04] which can result in lowering the user acceptance rate.

However, the parallel with the UCD approach is obvious: an understanding and appreciation of the users and their requirements [McN].

3.4.2 Testing

Constant and extensive testing is the heart of XP. It is mainly embodied by two practices: *Continuous integration* runs all existing automated tests whenever the code base is changed or extended in order to check if the changes caused any undesired side effects. Most of these tests emerge from *test-driven development*: first, automated tests checking the desired behavior are created; then, the actual behavior is implemented and can right away be evaluated with the tests. This usually is done only for pure behavioral code but can be extended to user interfaces: tests can check the expected behavior of an interface, and these tests can be run whenever the code is changed.

The end-user tests of UCD are a valuable source of test cases: an unexpected user action that caused a problem in the application can be replicated as an automated and continuously evaluated test to ensure that the problem, after solving it once, does not reappear. So UCD findings can augment and improve the testing Practice of XP and the XP testing practice can automate UCD test procedures.

3.4.3 Iterative Development

Both XP and UCD propagate an iterative procedure [GGB+03b] [Bec99b] of design and development [W3C04]. An XP project yields *small releases* (another core XP practice) on a regular and frequent base (usually a few months). Each release version is based on the previous one, incorporating new features and fixing bugs of the predecessor. Inside a release time frame, work is organized in "iterations" (usually taking one to four weeks). On an even smaller scope, many feedback-and-change iterations take place, especially in conjunction with test-first development and *refactoring* (the practice of changing source code in order to improve its quality without changing its functionality).

UCD also proposes a design–test–modify circle for developing user interfaces. The scope of iterative development in XP and UCD differs: releases and iterations in XP are mainly organizational units, while refactorings are just considered a development tool; on the other hand, UCD's iterative user interface refinement is a more explicit process as its involvement of external persons (the test users) makes it more complex. Nonetheless, iterative interface development of UCD fits well into the iteration principle of XP because both approaches are aware of the value and necessity of evolutionary development.

3.5 Usability Instruments

The following sections describe briefly the usability instruments which we used in our project. Also the differences and necessary modifications when employed in a agile environment are mentioned.

3.5.1 Personas

"Personas are not real people, but they represent them throughout the design process. They are hypothetical archetypes of actual users. Although they are imaginary, they are defined with significant rigor and precision." [Coo99, pp. 123-124].

Personas are a design tool based on the ideas of Alan Cooper, who released his book "The Inmates are Running the Asylum" in 1999, which is considered to be the founding work in the field of Personas [Coo99]. Since the invention of Personas, many scientists but also big companies have gathered interest in this approach.

The Personas method was developed as a tool for rising empathy for the end users in development teams and as a means for communicating peer group definitions [WTS+08, HMS+09]. Personas are archetypical figures - fictitious characters created as a tool to represent a typical user group.

Personas determine what a product should do and how it should behave. They communicate with stakeholders, developers and other designers. They build consensus and commitment to the design and measure the design's effectiveness. They describe the target user - her wishes, desires and application-specific aspects. They contribute to product-related efforts such as marketing and sales plans [Coo99]. Furthermore, they show the nature and scope of design problems [PA06].

If no Personas are defined in a project, the project members will always envision themselves to be the end-user, which leads amongst others also to communication problems [PA06]. Each Persona represents a peer group of users. With a detailed description of the Personas, every member of the project knows the main users and has a unified view of the target customers [Coo99, PA06]. The development of Personas is usually based on a very detailed design process starting with the gathering of background-marketing information, the design of Persona skeletons, their introduction, and a marketing campaign [PA06].

Personas are supposed to be applied not only once in a project, but throughout the entire duration of a project. The final Personas do not only exist on paper. Within a project they come alive through different scenarios and use cases that are built around the Personas [PA06]. Therefore, they can be seen as real people with lives and - if applied right - they will even feel real to the project members.

The advantages of Personas are at hand: As mentioned above, they allow unifying the picture of the target user for the project team, which allows for a more fluent communication [PA06]. Additionally, Personas make use of the "emotional mind" [ST05] of people which leads to a better focus on user-centered thinking within a project. "The user" as an abstract term is eliminated from the project and is replaced by people with names and faces, which, among other benefits, saves time because of shortened debates. To apply the Persona method, no existing processes have to be modified or changed; Personas can just be added to the project to focus more on the end-users.

Moreover, Personas allow for informed design and according to Alan Cooper, they enlighten the design process [Coo99]. Furthermore, Personas can also be used as an evaluation tool, such as walkthroughs [PA06]. As an archetypical figure, Personas can guide decisions about product features, navigation, interactions, and even visual design (among other factors) [NIA07].

In the agile development process, Personas can be integrated as so-called "Extreme Personas" [WTS+08], an approach to Personas that starts with the same activities like the classical Persona method: preliminary user groups are defined and Personas are modeled for them afterwards. In addition, the knowledge gathered in user studies is incorporated and the Personas will be refactored when the new knowledge suggests slight changes. If the found knowledge reveals that current Personas do not cover the insights, new Personas will be developed.

These actions make the classical Personas "extreme" by applying the XP paradigm of small iterative steps and refactoring - which is extending the Personas in this case. During the coding phases the developers pin the Personas beside the User Stories. Their first application is in the Planning Games (the phase where User Stories are created) where the Extreme Personas yield as reference representation of the end user. This is reflected in the User Stories which are written from the point of view of a Persona.

3.5.2 User Studies

User studies are the instrument for getting knowledge about end-users. The purpose of user studies is to uncover user needs, desires and contexts of use. In an agile

UCD process they can be used to develop new or refactor the existing Personas as well as in the User Story creation process where direct input can be derived from the studies [WTS+08, HMS+09]. In our project user studies were used in the form of laddering interviews and field studies.

Laddering Interviews: Laddering interviews are techniques that are mostly applied in the field of marketing and psychology. Nevertheless, recently it has also been applied to investigate user experience [PT07]. In a structured interview between two persons, the connections between attributes and their consequences are investigated from the interviewee's point of view [RO01].

The duration of a laddering interview can be – depending on the content – between forty-five minutes and two hours. The structured questions are employed to discover the respondent's beliefs, feelings and goals. In order to get familiar with the interviewee, a warm-up phase is needed and sessions are usually recorded.

Field trials: In field trials the developed products are tested by real users in an uncontrolled setting. The feedback of the users can be collected by applying various techniques. These techniques may include surveys, questionnaires, interviews, contextual inquiries, diary studies, etc.

3.5.3 Usability Tests

Usability tests are empirical studies that involve real users testing an application [HMS+09]. They can be considered as the most fundamental usability evaluation method [Nie93]. Contrary to field trials, usability tests take place in laboratory environments. Usability tests are conducted several times during product development to measure accuracy, user performance, recall-value, and the user's emotional response.

During the tests the users are observed while using the product. In some cases the users can be asked to think aloud and verbalize their thoughts to get a better insight into the user's mental model, as well as encountered problems. Other methods such as interviews can be combined with usability tests.

In an agile development process, the standard procedure of usability testing is too slow to cope with the changes in the application. Therefore, here an adaptation is required to fit the test and reporting period into one iteration to have the results present at the next planning. Additionally, as agile projects tend to change more quickly, more tests are suggested. To compensate for the increased frequency, the number of participants can be lowered and the results of different tests can be combined.

3.5.4 Expert-based Usability Evaluations

Expert-based usability evaluations are reviews conducted by experts [HMS+09], either in the area of usability, in the application area of the particular system, or

both. Two of the most renowned expert-based usability inspection methods are Heuristic Evaluation and Cognitive Walkthrough.

Heuristic Evaluation: Heuristic Evaluation has gained in popularity with Molich and Nielsen's introduction of ten heuristics to a wider audience [HN07, MN90]. The original heuristics have later been improved and adapted to different areas [Nie93] and are still frequently employed to evaluate different kinds of systems.

This evaluation method is considered an efficient analytical and low-cost usability method, which can be applied repeatedly during a development process. In general, heuristics can be considered as rules of thumb describing the affordance of a user to a system and are formulated more generally than the rather specific guidelines. They are recognized and established usability principles.

During a heuristic evaluation three to five experts (one expert at a time) inspect a system according to given heuristics. The found issues are categorized according to their severity after the evaluation is finished. Heuristics do not cover all possible occurring usability problems but because of the ease of application they can be employed very early in the design process - even when usability testing would not be possible.

Cognitive Walkthrough: A cognitive walkthrough is an analytical usability inspection method which was introduced by Wharton, Rieman, Lewis and Polson [WRLP94]. The main goal of a cognitive walkthrough is to measure the learnability of a system by detecting usability issues. Traditionally this evaluation method is conducted either by a single expert or a group of experts and novice users who put themselves in the place of a hypothetical user [HN07]. During the evaluation typical tasks are accomplished within the four phases of a cognitive walkthrough. Similar to heuristic evaluations cognitive walkthroughs can be applied very easily and early in the design process.

In an agile environment these methods can be used continuously and very early in the process. Because they do not depend on actual functionality, they can be applied to User Stories prior to planning and implementation which drastically reduces rework due to usability problems.

3.5.5 Lightweight Prototypes

Lightweight prototypes or Mock-ups show parts of the application without offering actual functionality. Only the layout and the visual appearance is presented. However, this makes it already possible to use usability methods like Heuristic Evaluation, Cognitive Walkthrough, and even user tests with real users. These prototypes are generally divided in paper prototypes and SW prototypes.

Paper prototypes come in various flavors ranging from hand-drawn rough sketches of the screen to photo-realistic images. The benefit of using paper rough mock-ups for the interaction design is that they can be designed and modified quickly. For simple interaction designs, a low fidelity paper mock-up suffices as a basis for further discussions and the implementation. An additional advantage is that it is

easier to criticize simple and rough mock-ups compared to ones which look neat and perfect from the graphic design perspective [STG03]. But for some features a high fidelity mock-up is required to clearly visualize the interface.

Software prototypes are actual applications without real functionality, but usually some kind of fake functionality is implemented to allow interaction. These prototypes can either be implemented in the same language as the final system or in a different one depending on the circumstances. However, if the languages are different care must be taken that the real implementation language offers the same possibilities as the one used for the prototype. For the extension of an existing application it is possible to use an existing system and make fake implementations of the new parts. The advantage of this approach is that the GUI is already implemented as intended, minimizing rework. The disadvantage might be that a due to special hardware or resource requirements of the application it is not possibly to freely distribute the prototype for testing.

In an agile environment, paper prototypes as well as software prototypes can be used. Paper prototypes are well suited for usage prior and during planning and can be used as part of the User Stories for additional clarification. Software prototypes can be generated during development as part of the application to get feedback from the customer. Then they are completed by incorporating functionality. This corresponds to the XP motto "*Fake it till you make it*".

3.5.6 Automated Usability Evaluation

The idea of automated usability evaluation is not new. Basic research goes back to the early nineties [BCS93, HHH92]. In the year 2000 the state of Automated Usability Evaluation (AUE) is still described as "quite unexplored" [IH00].

AUE offers usability support through specialized tools. Therefore, developers can be supported by automatic inspection throughout the development phase of a project. An example of such kind of automatic evaluation is log-file analysis [AS07, HL01]. Here the generated data helps identify paths and execution time in order to detect problems.

Another example for AUE is the NIST-Web-Suite[3] that allows for an automatic code-based analysis of websites according to 12 design guidelines. The WAUTR-project[4] (Automatic Usability Testing environment) is a first attempt to support usability experts with a set of different tools.

The availability of user-generated data already during the development is one problem of AUE. The simulation of the final users [Abd04] or specific aspects (e.g., gaze[5]) is one possibility to solve this issue. Nevertheless, also this approach requires actual user-data.

When there aren't any users available, code-analysis is suggested as the next best solution. The code is used to calculate usability factors and give input on usability based on design guidelines (e.g., the ratio between text and graphics on a website,

[3] http://zing.ncsl.nist.gov/WebTools/
[4] http://wauter.weeweb.com.au/
[5] http://www.goodgaze.com/ggx/, http://www.feng-gui.com/

the number of links, the use of colors, etc.) [IH00, IH02]. This approach is currently mainly used in the web area. WebTango is one of the best known tools in this field of application [Ivo00]. It calculates usability metrics from HTML code and the evaluation is based on a statistical model of the website usage.

An approach that goes even further was developed by Maysoon Abdulkhair [Abd04], who has implemented agents that are able to learn from user-behavior. The underlying statistical model allows the agents to detect user preferences, learn them and use them to evaluate websites.

The reverse-engineering of the structure of a website was used by Paganelli and Paterno [PP02] in order to find potential usability problems with their tool "WebRemUSINE".

Currently, the main target group of such kind of tools is experts in the area of human-computer-interaction. The transition towards developer-based tools for AUE is currently in progress [WTS+08]. These tools would then be able to continuously check for usability issues, even during development while the code is being written. Although gaining in popularity, automatic usability evaluation can rarely be found in commercial development environments. The most renowned product of this kind is LIFT[6], which is comparable to WebSAT. Additionally, LIFT integrates the GoLive, FrontPage and Dreamweaver development environments.

3.5.7 Extended Unit Tests

The idea of Extended unit tests originate from Automated Usability Evaluation (AUE) [HMS+09]. It is an natural extension of the mandatory unit tests which XP demands. Usability evaluation tools should be included in the automated unit test procedure to get not only a feedback about the correctness of the program, but also about its usability.

Since most of the current approaches to automated usability evaluation tend to focus on multimodality [PPS06] and mobile devices [WHS+02], the existing tools have a big disadvantage for our project: most of them are isolated solutions which are - as mentioned before - solely designed for HCI experts. Hence, they hardly integrate seamlessly into the existing development processes.

3.5.8 XP/UCD Combination

Usability is essential for the success of an application and measures have to be taken to include usability instruments and practices into the development process.

The UI design process according to UCD is largely beneficial as it provides feedback [GGB03a] which is used for the system's functional requirements. The assessment of each feature from the users' perspectives influences the whole development process of the application and addresses the problems which arise when the system requirements are gathered only by discussions with stakeholders [JA04].

The UCD approach fits well into the XP process because of the many overlapping principles (iterative development, end-user incorporation, testing) of both method-

[6] http://www.usablenet.com/

ologies. As XP is also a lightweight process that puts very little administrative overhead on the developers, extending XP with additional practices is much easier than for other, more restrictive methodologies.

Although XP and UCD are two different methodologies, both focus on the user. Due to this same main focus both methodologies can be integrated very easily [GGB03a]. But XP lacks in knowing their true users and UCD lacks of a flexible and adaptive development methodology that lasts throughout the entire project [Tox05]. Therefore, the integration obviously results in complementing each other and the resulting process has the advantages of both worlds and at the same time minimizes the deficiencies of both methodologies.

Many of the usability instruments can be used readily inside an agile environment making it easy to augment the XP methodology with additional usability practices. However, slight modifications have to be made to these instruments to fit the need for frequent and quick feedback.

On the business side we see the need for a more elaborated process on how to include different stakeholders and their input. We assume that the more stakeholders get involved the higher the need for structured inclusion strategies for all stakeholders will become. Research on these inclusion strategies will be necessary to ensure that the input of each stakeholder is treated the right way.

3.5.9 Potential Problems

XP and UCD fit together very well from their ideas and their process. However, there are several issues which can prevent the successful integration of HCI instruments into an XP processes [WTS+08].

Ad-hoc Input: Because of the short release cycles software engineers would need ad-hoc usability input during development. In practice usability input is not given ad-hoc but after longer periods (one to two weeks average). Such timespans are not acceptable for most XP practitioners.

Cultural problems: Software engineers on the one hand and HCI experts on the other hand come from different domains with different attitudes, approaches, backgrounds, and even different ways to express themselves while communicating. The XP process requires tight cooperation in teams, which reveals differences between engineers and HCI experts very quickly: engineers have a technical approach to software development whereas HCI experts mainly have a psychological background, hence taking a cognitive view on software development. As these differences can lead to problems, methods to prevent these have to be integrated into the collaboration process.

Technical Focus: By its genesis, unit tests in XP environments are designed for technical testing. Hence, the focus is on technical functionality – ignoring usability issues. This means that the technical view of testing has to be expanded by HCI approaches and means.

On-site Customer Representative: From an HCI point of view the inclusion of customers is a step into the right direction. But the Manifesto for Agile Software

Development[7] does not clearly demand end-users as customers. We expect deficits in usability if it is not clearly stated that end-users have to be part of the process. Developers need to have a clear picture of the humans they develop for.

Awareness: In order to successfully include usability and user-centered design in an XP process the developers need to have a basic understanding of usability issues.

[7] http://www.agilemanifesto.org

Chapter 4

M3 Project Structure and Context

Mobile computing is leading a revolution. Our lives are changing at a pace never experienced before in human history. A wide variety of applications for mobile phones is available at the moment. Still, there are not many of them which utilize the available bandwidth and at the same time are accepted by the users.

Studies show that multimedia – Audio and Video (AV) – consumption is on the edge to become one of the next killer applications for mobile devices [DLH06]. User behavior in consuming AV is changing. Traditional broadcasting is losing more and more audience because online and mobile AV intrudes heavily into this area. A recent report states that 43% of Britons who watch video regularly from the Internet or on a mobile device are now watching less TV than before [BBC]. Clearly, it is in the interest of broadcasting companies to adapt to these changes in user behavior and invest in these new technologies. For these companies, one of the major advantages of mobile phones compared to other devices is that they can charge for their services easily and directly, as the existing infrastructure can be reused. At the same time, customers are given the flexibility to access rich multimedia content from anywhere, at any time. A market survey showed that consumers are interested in using this technology and are ready to pay a realistic price for such services [CW07].

We wanted to research the Extreme Programming (XP) methodology and the inclusion of usability methods in a real world context. Therefore, we needed a project which is targeted towards business success and real users and where we can apply the agile principles. We found these circumstances in the realm of mobile phones. Here you have a substantial growing business, quick changes in technology and demands, as well as a huge number of users. Also the usability demands are high because of the different skill and technical level of the users as well as the different environments mobile phones are used in and the limitations of the devices 3.1.

We developed an application that enables a user to perform content-based search for AV material and play it on a mobile phone. The application enables a user to search not only in the meta-data but also in the spoken words of the AV clips. This content includes radio and TV archive material, like documentaries or other recordings of historical, political and cultural importance, discussion programs, movies, music videos, audio books, etc.

The application addresses not only the objectives of the content providers but also keeps in mind the emerging functional and cognitive needs of the users. It was designed keeping in mind the importance of usability aspects and also allows social interaction of users by providing Web 2.0 features to encourage community

building [HLM+08b].

The scientific goal of the project was the analysis of agile software development methods, particularly XP, the inclusion of usability methods into XP, and to devise a usability test procedure for mass applications on mobile devices with emphasis on User Centered Design (UCD) and iterative user-interface design.

The project started in summer 2007 and ends in 2010. It is part of the Softnet projects [www10b]. This gave us access to industrial partners which helped us on the technological and technical side as well as provided a real business context.

4.1 The Softnet Project

Softnet Austria is a private research association cooperating with business and university partners to conduct and promote applied research in software engineering. Softnet Austria deliberately operates as an intermediary between research institutions and companies. The network is targeted on quality assurance and improvement, and at the development and deployment of novel techniques in software engineering. Besides of conducting applied research in next generation software engineering, Softnet Austria provides an institutionalized access to Austria's software engineering scene.

Softnet Austria is an industrial competence network within the so called K-net program of the Austrian Federal Ministry of Economics and Labor in terms of private non-profit association. The competence center organizes its research and innovation work in terms of working groups under academic supervision. In turn, the working groups are assigned to the competence fields of engineering methodologies and models (SOFT-T&M), and web engineering and user interfaces (SOFT-WEB). [www10b]

The main focus in our project (Softnet project 7) is to evaluate and adapt agile SW development processes and their interaction with usability processes for small and medium sized companies in Austria.

4.2 The M3 Core Team

Our project is a research project with six PhD students as full-time regular members, and one PhD supervisor. The full-time team members having different roles, a different experience level in SW development, and a different social and cultural background. Five of the PhD students are developers and one is a business student. The business person, having a degree in business science, deals with project partners, is responsible for marketing, and acts as on-site customer and product manager. On-site customer is a role defined in the XP process which sees the customer as part of the project team and demands that the customer is always present for clarification and testing purposes, substituting huge requirements documents. He shares these responsibilities with the PhD supervisor who is not always available because of other duties. As he has a professional experience in team mediation he acts as a mediator in team discussions.

Three of the developers are from Europe and have already worked as full-time SW developers in industry for some years. The two other developers are from South Asia, have an academic background and teaching experience, but no prior industrial experience in SW development.

Our project has partners from industry and the business goal is to produce a commercial successful application and to start a company. The project partners come from various domains, including User Interface (UI) design, usability research, telecommunication, content providing, and hardware infrastructure.

For business related activities the customer communicates with the partners. Otherwise the communication is done directly by the developers. Developers and customer also directly communicate with the engineers of our partner usability research center regarding usability issues. This communication is done throughout the whole development cycle: From User Story creation (mainly by the customer), over implementation issues (mainly by the developers), to the final usability assessment of a story or the whole system (mainly by the customer). The usability engineers working for our project are also active in UCD research together with the team.

4.3 Our Internal Partners

Our internal project partners are part of the Softnet Austria project and support us with knowledge as well as resources. The come from various sectors of the industry.

CURE - Center for Usability Research & Engineering [www10a] was founded 1996 as a working group for Human-Computer Interaction (HCI) at the university of Vienna by Univ. Prof. Dr. Manfred Tscheligi and Dr. Verena Seibert-Giller and is since 1999 an independent Usability Research institute.

As our partner, CURE supported us with the knowledge about user interface design and usability. First of all they provided us with a dedicated usability engineer which was included into our development process although not physically present on site. In the filed trial CURE took a major role during the planning phase and conducted user and diary studies during the trial. CURE also conducted frequent expert reviews of our application, conducted a usability test with an early version of our system in their usability lab, and provided the user profiles and Personas for our development.

E-NOVATION Gmbh [wwwa] is an expert in the field of Content management systems and aesthetic design. While usability is mainly targeted to identify and remove obstacles for the user in the interaction with the system, aesthetic design is meant to improve the visual appearance and make it more appealing. They provided us mainly with suggestions regarding colors and visual elements. One major contribution form E-NOVATION to our project was there excellent business knowledge which helped us a lot during negotiations with potential customers.

Kapsch CarrierCom (KCC) [wwwc] provided us with facilities and resources. They financed additional resources and provided the hardware and infrastructure to host

our system redundantly. Also on the business side KCC provided help by incorporating their major customer Mobilkom Austria into the project which was our carrier.

SAIL Labs Technology CORP [wwwh] is a leading global provider of speech recognition technology. They provided us with the software to enable our system to generate speech transcripts from AV clips and supported us during the installation and integration.

4.4 Our External Partners

Mobilkom Austria [wwwe] is the leading service provider in Austria for mobile phones. Together with mobilkom austria we organized a big user trial and applied for the Austrian Muitimedia Staatspreis [wwwf]. For these events mobilkom provided the necessary phone hardware. Additionally, they recruited the 185 trial users from there pool of customers.

ORF (Austrian Broadcasting) [wwwg] is Austria biggest broadcasting company. The ORF acted as our main content provider during the project and especially during the trial.

Krone.tv [wwwd] is part the Kronen Zeitung, Austria's largest newspaper. Krone.tv is another content provider for our project which was acquired at a later stage.

Chapter 5

The M3 Application

Our system offers a lot of flexibility for searching and consuming AV content. The M3 application allows it to stream video clips over the standard infrastructure of cell phone providers to mobile phones. It allows to search in meta-data (e.g., title, descriptive text) as well as in the transcribed speech. Media streams can be played, interrupted and resumed on the mobile phone. AV content is stored in a database containing meta-data (title, summary, duration, etc.) as well as transcribed speech from the clips which is generated by the use of reliable speech transcription techniques [BRWH00]. This makes it possible to extend the search to words spoken inside a clip. The system's User Interface (UI) and media delivery are based on standard web technology.

Besides this, community-building features are provided in order to encourage user interactions, keeping in mind the social interests of users. The aim is to build a community platform for mobile phone users where they can share their views and opinions about the provided AV content. Moreover, feedback from the community will reflect the current trend of multimedia consumption.

The personalized approach of the system makes it possible to implement user-based recommendations. To achieve this, data is collected by means of two information acquisition models. The interactive model is based on user ratings while the behavior-based model relies on collected usage data. Information about the clips consumed and the duration of the consumption provides information about their interestingness. Additionally, the collecting of usage data provides continuous feedback, enabling constant improvement of the system.

Special emphasis was put on usability of the system. To ensure this, a number of usability instruments were applied and incorporated into our agile development process (see Chapter 3).

5.1 Related Applications

This section provides an overview of different AV search and streaming applications. They can be categorized by means of features they offer, particularly speech recognition which is used to enhance the meta-data for advanced search and streaming on mobiles.

Mobile YouTube [You] and MobiTV [CW07] allow searching for content and stream it on mobile phones. Mobile YouTube has the big advantage to have Google as parent organization but the content is very limited and consists only of user-contributed

material. In comparison to that MobiTV offers different real-time worldwide TV channels streaming without search functionality, but for streaming on a mobile phone, client-side software is required.

In contrast to this, Blinkx [Bli], TVEyes [TVE], and EveryZing [eve] perform speech recognition in order to enhance the search with additional meta-data but do not provide mobile phone streaming. Blinkx provides user-contributed content as well as content from broadcasting companies. The content offered by TVEyes is restricted to news material and the content of EveryZing is restricted to web based content.

Other applications offering online video search and streaming are JumpCut [Jum] and Joost [Joo]. Neither has the feature of speech recognition or streaming on mobiles. Jumpcut has the big community of Yahoo users but is limited to user contributed content while Joost provides more content but is based on peer2peer technology requiring its own client-side software and is still in its beta phase.

This comparison shows that there is no application which offers both features at the same time like our application does.

5.2 M3 Usage Scenarios

In daily life, many people are at work or school at day time. They have no time to watch TV or listen to radio programs, because of the schedules set by the broadcasting companies. Time-delayed, played-back, and individually-delivered AV on mobile phones provides a new platform taking radio and TV into the street, car, public transport, waiting room, park, and virtually any other location. This type of application creates additional audiences eager to access multimedia content during new prime times set by themselves, that is, in their commuting periods or other idle times. The basic idea of such a system can be illustrated by the following sample usage scenarios.

5.2.1 TV Archive for Subway Riders

- A commuter in the subway searches for “Fernando Alonso”.
- The system matches each word of the textual search query with the positions it occurs in each AV clip.
- The system presents a list of clips in which the name “Fernando Alonso” has been mentioned, sorted by temporal occurrence and relevance based on content, e.g., how often the name was mentioned.
- The user selects one of the presented entries.
- The system’s media server delivers the selected clip to the user’s mobile phone.

5.2.2 Radio Archive for Car Drivers

- A user listens to the last sentences of a radio broadcast about the “European Constitution”. The user still has to travel with the car for some time and

therefore searches for the keyword “European Constitution”.

- The system lists a number of related news items, interviews, and documentaries.
- The user selects the desired topic.
- The hands-free set of the mobile phone plays back the selected material through the car’s stereo.

5.2.3 Media Recommendations for Users

- A user wants to consume some AV content but has nothing particular in mind.
- The user asks the system for recommendations.
- The system generates recommendations based on the user’s stored preferences and on other users’ recent behavior. A short description of each item is also provided.
- The user selects an item and plays it on the mobile phone.

5.2.4 Interactive Video

- A user sees an interesting scene in a clip and wants to comment it.
- The user enters the comment in the system.
- Another user watches the same clip later.
- When the scene comes she sees the textual comments of the previous user on the screen.

In the current version of the system all scenarios are fully implemented beside scenario 5.2.2 which was dropped for the moment. This was due to the fact that the system currently focuses on video clips which are not appropriate for car drivers.

In all scenarios it is possible to resume at the previous location if the user stops the media stream. This feature is unavailable with regular broadcasting or streaming systems. The user of this system has more flexibility for consuming the AV content. As further improvement, it is planned to allow this resumption at any time, even weeks later. This is particularly important because of the short continuous viewing or listening periods. For example, while commuting, interruptions and (possibly much) later resumptions will be the regular case.

Such behavior is rather uncommon for AV consumption so far, especially for viewing video. But it is not so much different from the way a book is read, having breaks between reading periods. Thus, it seems plausible that users will be willing to switch to this new way of listening and viewing AV content with interruptions as it brings them the convenience of being able to decide what to consume in a just-in-time way, independent of place and time.

5.3 System Architecture

For the implementation of the system, standard components and technologies have been used. The AV clips are stored on a hard disk, while the additional data is stored in a database containing transcribed speech from the clips, additional meta-data (title, summary, etc.) where available, and user contributed data like comments, ratings, etc. The media delivery is based on standard web technology. This enables people to use this service with almost any modern mobile phone.

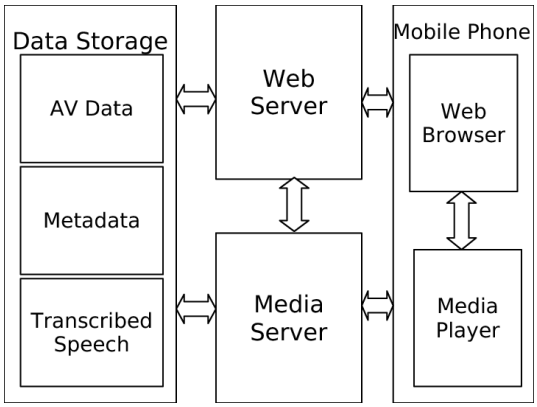


Figure 5.1: System Component Diagram.

Figure 5.1 presents a coarse grained overview of the components of the system. The server-side system comprises data storage, a media server, and a web server. The client side is represented by mobile phones with multimedia and web browsing capabilities.

Data Storage

The core of the system is data storage, containing the AV data, the meta-data, and the transcribed speech. The AV content is passed through a number of steps, starting with content logging using segmentation of the videos into a set of coherent units or shots. All video shots are properly logged, thus generating a Table of contents for the video. Each shot is tagged with information through which it can be uniquely identified. The audio parts of the shots are transcribed by passing them through a series of procedures, beginning with automatic speech recognition and ending with speech transcription [BRWH00]. This is done using a system from our partner Saillabs Technologies (see Section. 4.3).

The transcribed audio parts are stored in a database together with additional meta-data from other sources such as webpages, manual descriptions, etc. These meta-data enables additional search possibilities within the database.

The AV clips are transcoded to a compatible format (MP4 with H.264) in a pre-processing step. They are stored directly on the hard drive and have a reference in the database.

Web Server

The web server manages client requests and hosts the user-interface for browsing and searching in the database. Because of the diversity of available mobile phones, the system is web-based and is accessed through the pre-installed web browsers of mobile phones. Therefore, the challenge of having to deal with many different client devices is alleviated.

Media Server

The media server is responsible for delivering the actual AV content to the client and supports advanced user interaction. This includes stopping, fast forwarding, skipping parts of the AV content, resuming later, and replaying a sequence. In this project, the open-source streaming server Darwin [App] is used.

Mobile Phone

The system does not require specific client software to be installed on the mobile phone. Instead, any state-of-the-art HTML browser and any video player capable of receiving AV streams can be used. Because of the constant evolution of mobile phone technology, the focus of the application development process is on today's state of the art technology.

5.4 Community Building Features

There are many - sometimes overlapping, sometimes contradicting - definitions of virtual communities. Although the phenomenon of virtual communities seems to be rather new, we can use a definition from 1993 by Howard Rheingold: "Virtual communities are social aggregations that emerge from the net when enough people carry on those public discussions long enough, with sufficient human feeling, to form webs of personal relationships in cyberspace" [Rhe00]. This casual formulated sentence defines essential points: We deal with social phenomena which occur in the context of the internet with the primary goal to establish a relationship between humans [LL08].

Web services and sites life and die with their users. There is the observation that sites wit many user are more likely to accumulates more users and grow, and at the same time sites with few users will shrink even more. This is called the "Matthew effect" named after a line in the gospel of Matthew: "For to all those who have, more will be given, and they will have an abundance; but from those who have nothing, even what they have will be taken away." Matthew 25:29. Therefore, it is important for websites to be attractive to users. What attracts new users, beside the content, is the already existing community.

In a study done by our partner cure where they analyzed the intrinsic motivations of people to consume and produce multimedia content, they found that "Exchanging and showing content with the device is a task performed in order to maintain social relationships." and that "The process of exchanging content and entertaining others

is experienced by users as pleasurable." [LWST08]. This supports findings form O'Hara [OMV07] who reports that "sharing experiences" is an important reason to use mobile video content, as well as Nettamo [NNH06] who reports that the exchange of content is motivated by the need for social interaction and inclusion.

To address the social interaction aspects of users, our system provides different community-building features. Users of our system can rate a clip and give comments to clips which support others and allows discussions and opinion sharing. Furthermore they can insert comments directly inside a clip which can be seen by other user when they watch the particular part. This allows a more direct and scene specific discussion and interaction.

5.5 User-Based Recommendations

Web-based companies already use recommendation systems with great success. Amazon, for example, has millions of customers. Seeing the benefits of recommendations, Amazon has developed its own technique called "item-to-item collaborative filtering". Their customers regularly take advantage of these recommendation facilities when making their purchases [LSY03].

The personalized approach of our system makes it possible to implement user-based recommendations. The unique identification of a user is necessary for accounting purposes, implying that a user profile has to be managed by the system. This profile is augmented with additional data which is used for recommendations to the same and to other users. The data is collected by means of two information acquiring models, the interactive model and the behavior-based model.

5.5.1 Interactive Model

The interactive model is based on user ratings. After users finish consuming an item, they are able to rate it according to their liking and preferences. Information about clips that users consumed and their respective ratings are stored in their individual profiles. The rating of a clip in each user's profile affects the overall rating of the clip in the database. The individual ratings are still traceable. For more personalized recommendations, ratings of similar user groups can be combined.

5.5.2 Behavior-Based Model

The behavior-based model is applied by collecting usage data and information about the clips consumed in the users' profiles. This is used for user-specific recommendations. Clips other users played which also played the current clip are presented as recommendation to the consumer.

To enhance the current recommendation algorithm and develop a more consumption oriented recommendation system, it is planned to store also the duration of the consumption. If many users stop the same clip after a short time, this clip is most likely not very interesting. Of course, this equally depends on the overall playing time. Therefore, a ratio measure will be used for clip rating. The system will take

into account that users are allowed to stop and resume clips at any time which can influence the measurements. Alternatively, it will be possible to consider only the behavior of a specific user group.

5.5.3 Model Combination

These two models are combined when generating recommendations. For the system, user ratings are more important than usage data. However, ratings may not be available for every item. In this case only behavior data is used.

At the moment there are three recommendations given. On the main page the top rated as well as the most recent clips are displayed, while when viewing the details of a clip, an "others also watched" section is provided which shows clips watched by other users which watched this clip ordered by rating and recency.

Additionally, different scopes for the rating mechanism are planned for the future. The user will be able define their own default recommendation setting by user preferences. On the one hand all users can be considered, and on the other hand just a specific user group can be considered. This will results in different recommendations. Furthermore, the changing preferences of users will be taken into account by adding a time-descending weighting factor.

5.6 Implications

An attractive feature of the system is the possibility to target advertisements more precisely. This feature is useful for companies wanting to address specific user groups. Additionally, users benefit because they receive only advertisements related to their interests. For example, Google's Gmail is using this technique for advertising purposes on its popular mail accounts. The large user base of Gmail is a valuable target for business. The advertising is tailored to users' mail contents. Gmail also offers the possibility to use mobile devices [Goo]. It is expected that this trend will continue as Google has purchased YouTube. The advertising and search capabilities are, or will soon be, extended to video content [McL07].

The feature of collecting additional user data provides continuous feedback, enabling constant improvement of the system. By recording this information, valuable data about how the user is interacting with the system is obtained. This allows reacting quickly to new usage patterns and needs as they arise.

Our system allows at the moment the addition of text which can be displayed at specific locations inside a clip. As this is thought as a way of user to user communication, the same concept can be used for advertisements inserted at well defined locations.

Chapter 6

Evolution of the M3 Application

The m3 application development started in May 2007 and lasted over two years. During this period a number of releases and changes have been made to the application as well as to the original concept. This section gives a brief overview of the m3 development during this period and also shows how the current application looks like.

6.1 The Initial Application Version

The basic goal we wanted to achieve with this version of our application was to set up the necessary infrastructure to be able to stream a video from a streaming server to a mobile phone.

To set the minimum requirement for the M3 0.0.0 we choose 3GP and MP3 as video and audio format. 3GP as container format for mobile video was at that time state of the art and is supporting H.263 and MPEG-4 part 2 as video formats and AMR and AAC as audio formats.

We also did some research on different streaming servers and finally decided to use the Darwin streaming server 5.5 which is the open source implementation of Quicktime's streaming server. To use this server we have to pre-process ("hint") the video files in order to help the Darwin server to identify the audio and video tracks.

For the purpose of hosting an HTML page containing several links to videos stored in a folder of the Darwin Streaming Server we used Apache Tomcat 6.0.0.

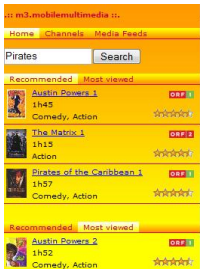
The Darwin streaming server and the Tomcat installation were hosted on the Windows Extreme Programming (XP) platform and were accessed from the mobile via a Wireless LAN Router establishing a wireless connection acting as HotSpot access point.

The mobile phone used for testing purposes was model Nokia E65.

In this first release we reached our self defined acceptance criteria, which was to stream a video from a streaming server to a mobile phone after selecting it from an HTML page. Thus we had implemented the minimal application which covered the most crucial technological parts of the system.

(a) Release 1

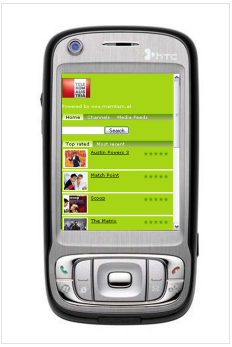
(b) Release 2



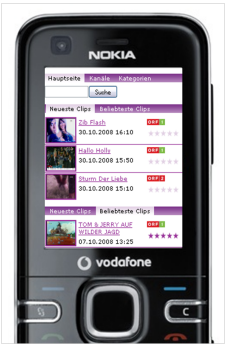
(c) Release 3

(d) Release 4

(e) Release 4



(f) Release 5



(g) Release 6

(h) Release 7 (Trial version)

(i) Current version

(j) Current version displaying in-video comment

Figure 6.1: Evolution of the m3 application.

6.2 Release 1

In this release we added one basic but important feature to our application, namely to search for clips on a mobile phone. The Darwin streaming server and the Tomcat web server installation were moved to two different Linux machines. For testing, we acquired an additional phone, namely the Nokia N95, which was at that time the new state of the art phone for mobile web access and multimedia.

6.3 Release 2

The two major goals of this release were to set up a sound technical infrastructure for the development process, and to include additional features as well as demo content required for external presentations of the application.

While until now we were limited to our WLAN connection, this release allowed connecting to the m3 application via UMTS. This enabled us to access the application over the regular mobile phone infrastructure, which was an important step towards our goal and necessary for external business presentations. We also changed the web technology of our application from html to the popular "Java Server Faces" web framework.

On the development side we established a Continuous Integration and a Nightly Build process using "CruiseControl" which periodically checks for changes in source code in the application repository and then re-builds and deploys the application.

As a major business-related goal we assembled demo AV content required for the external presentations of the application. To display these items, our Channels page simulated the program browsing page of the ORF website, providing a similar user interface as well as some original ORF AV content.

The application contained five main features, some of which were not fully implemented but had been included into the application for demonstration purposes. The user interface consisted of three screens. The application main screen which provided "Simple Search", "Recommendations", and "Most Viewed Clips", the "Channel Browsing" screen, and the "Media Feeds" screen.

6.4 Release 3

The major goal of this release was to prepare a presentable version of the application for the planned presentation to the ORF. Therefore, the user interface of this release provided similar functionalities as the ORF homepage. The design provided by our partner E-Novation (see Section. 4.3) was incorporated with small modifications. Otherwise the screens of the application did not change.

6.5 Release 4

The major goal of this release was to prepare a demo system for people from the ORF to allow them to try our system for a certain period of time. Also an additional

demo version for another potential customer was produced.

To accomplish this, we introduced a production and a development server, an additional PC for the Saillabs speech recognition system, and wrote two additional programs. A data feeder was created which downloaded the ORF schedule from the ORF RSS feed to get additional meta-data and a converter was written which processed clips provided by the ORF in an agreed format on a dedicated ftp server.

This data consisted of the clips and cutting files including information about the subtitles and timings. The program converted and cut the clips, made the speech recognition (transcription), extracted thumbnails, and copied the data to the production server and the database.

The programs were supposed to run automated at regular intervals. However, here the first problems appeared. As the data was manually provided, the agreed naming convention was not always obeyed which caused the converter program to fail. Also the time information in the cutting files did not always have the same format. Additionally, the summertime switch led to failures of our data feeder because for some reason the time format of the RSS feed was changed two times.

The user interface for the demo was the same as in the last release but some of the missing functionality was now implemented. On the detail page the users could rate and give comments and these ratings and comments were displayed. For those comments the user could choose an avatar from a list of pictures. Also Recommended and Most Recent Clips (formerly Most Viewed) had been fully implemented.

6.6 Release 5

The major goal of this release was to negotiate about the conception of a field trial as a real-market situation and integrate our partners into the XP process.

The user interface had evolved according to the usability inputs from our project partner CURE, e.g. the calendar component for selecting the date of interest on the channels page was redesigned in order to ensure that only valid values could be entered (i.e. disallowing selection of future days).

The search interface of the application had been improved as it represents a key factor in the usability of the system. Therefore, it was now possible to preselect a certain channel for a more sophisticated search process.

The Web 2.0 features have further been improved by establishing an SMS-based communication channel between users in order to allow clip-based recommendation, adopting viral marketing strategies.

6.7 Release 6

The major goal of this release was to prepare the application for the now fixed trial commitment of the Softnet project 7 group with ORF and mobilkom. The final usability contributions for the real trial application were implemented and the infrastructure for the trial was set up. The user interface was not developed further but optimized in certain aspects.

Unfortunately there was some additional work involved which did not concern our core application and took a great deal of our resources. In contradiction to previous assumptions, the ORF was not capable of supplying us with sufficient content for the trial system. Therefore, we had to implement a 24/7 recording system producing video clips for the channels ORF 1 and ORF 2. We also incorporated the program information (program name, start time, duration) from the ORF's RSS feeds which are publicly available on the Internet. This data was used to implement a cutting mechanism which was able to extract video clips from the continuously streaming TV signal. For this purpose we had to set up a completely new infrastructure consisting of a satellite dish and receivers for ORF1 and ORF2.

6.8 Release 7

The major goal of this release was to prepare and conduct the trial project of mamtam (the official name of the m3 application) together with mobilkom and ORF.

We further developed the 24/7 ORF recording system. The hardware setup was adapted for the demands of the long-running trial project: two recording machines (for ORF1 and ORF2) perform the recording of the raw MPEG signal and the cutting and encoding to MP4 clips; two additional machines perform the speech-to-text analysis of the produced clips.

We changed the mechanism for accessing the video signal from satellite to DVB-T. Apart from the video stream, the DVB-T signal also comprises an EPG (electronic program guide) signal. We utilized this EPG signal to extract program information and the exact start and stop times, allowing us to perform a much more accurate cutting of single programs than the unreliable RSS feed. The ORF's RSS stream is still used to define a program's genre and description (these fields are missing in the EPG data).

Additionally, we implemented a prototypical recording system for radio (Ö1) programs. The Ö1 content was not available to the trial users and was only used for internal evaluation and demonstrations.

The application's performance and stability were significantly enhanced before and during the running trial project. No new features were added to the application; instead, the existing features were stabilized to ensure a sound trial. Likewise, the user interface was only changed slightly in order to unify the look and feel of all pages.

The application was instrumented with additional "tracking" log messages in order to allow a-posteriori evaluation of the user's navigation behavior, usage of certain features (e.g. search, channels), clip consumption duration/frequency, and the preferred content types (i.e. the genre). The data produced by this tracking subsystem, together with the outcomes of trial questionnaires and usage diaries, represented the major outcome of the trial.

6.9 Release 8

During the trial, which started in December 2008 and lasted until May 2009, only bug-fixes and small changes were made and no new features were introduced to not distort the results.

6.10 Current Version

Parallel to the trial version, we developed our main branch further and added some new features concerning Search and user interaction.

6.10.1 Application Main Screens

(a) Main Page (b) Archive Page (c) Category Page

Figure 6.2: Application main screens.

Figure 6.2 shows the main screens of the m3 application. Each of these screens allows immediate access to the search function which is a main feature of our application. The prominent search box communicates this to the user. An important feature of this search box is that a once entered search query is stored through the pages, which means that it can be easily repeated under different conditions, e.g. restricted to a specific category. The equally visible check box, which toggles the search in the transcribed speech, communicates also this feature clearly.

The main page in Figure 6.2a displays two sets of clips for recommendation, the most recent and the most popular ones. Each of this sets has an own page which can be accessed by the use of a more link at their bottom (not visible in Figure 6.2a).

The category page (Figure 6.2c) displays only clips from a selected category. This category can be selected from a set of predefined categories with a drop down menu. Inside this views the ordering of the clips can be changed from a time based ordering (newest clips are displayed first), to an alphabetical ordering. On this page, the search functionality performs the search only in the selected category, allowing an

easier selection.

(a) Archive view of one day (b) Calendar expanded

Figure 6.3: Archive view with time selection.

The Archive Page (Figure 6.2b) allows the user to browse the archive, either as a whole or by a specified date. Also on this page it is possible to change between a time ordered and an alphabetical view. Additionally, it is possible to specify a single date with a built in calendar.

On each of the main pages, the clips are presented together with additional information such as thumbnail, title, user rating (stars), channel (logo), duration, and broadcast time. A clip can either be directly played by clicking on the thumbnail image, or additional information can be displayed be clicking on the title which opens the Clipdetail page.

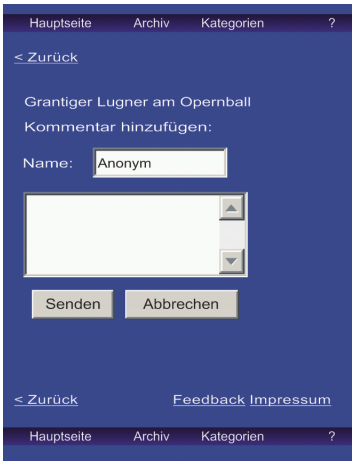
6.10.2 Clip and Search Pages

The Clipdetail page (Figure 6.4a) displays the description of the clip (if available), recommendations based on other users' behavior, and comments on this clip. It also allows the user to rate the clip and to leave a comment himself (see Figure 6.4b).

Figure 6.5a shows the result of a search. The result list presents each clip together with a small sample of text with the found keyword highlighted. Also the total number of hits in this clip is displayed. Clicking on the thumbnail plays the clip and clicking on the title displays clip details as described above.

Additionally, it is possible to click on the text sample which opens up a Searchdetail page as shown in Figure 6.5b. This page displays the basic clip information as well as the search results in the description and transcription. The hits in the description section are presented displaying the whole description and highlighting the found keywords. The hits in the transcription are shown as a list of text samples with time information and highlighted keywords and as timeline with markers. It is possible to immediate start the clip directly from these positions either by clicking on the marker on the timeline or on the time information in the list.

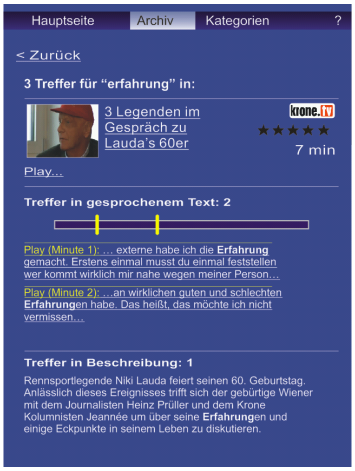
(a) Detail Page for a clip



(b) Comment entry for users

Figure 6.4: Clipdetail page features.

(a) Search result page

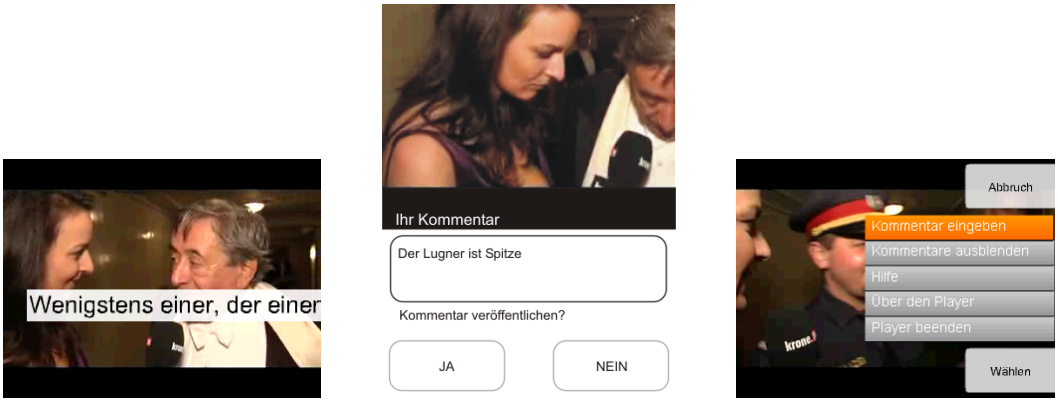


(b) Search detail page for one specific clip

Figure 6.5: Search specific pages.

6.10.3 In-Video Comments

In our latest version in-video comments were introduced. In-video comments are comments displayed during video playback at particular locations in clips (see Figure 6.6a). Users can enter these comments and other users who watch a clip after an in-video comment was entered will see it at the same location. For this feature we included a custom made device specific player developed by Grizly [wwwb] into our application. This player allows to enable or disable in video comments as well as to enter them (see Figure 6.6c and 6.6b).



(a) In-video comment in clip

(b) Entering in-video comment

(c) Player features

Figure 6.6: In-video feature.

6.11 Removed or Disabled Features of Earlier Versions

During the development of the project, some features have been removed again or have been only partly developed. As they have been mentioned in the text, I will provide here a short description of the feature as well as the implementation status and the reason for removal.

Channel Page:

"Channel" allowed to select different broadcasting channels, e.g., ORF1, ORF2, Ö3, etc. The channel could be selected similar to the category page described above and the broadcast date selection was similar to the Archive page. Additionally, each day was divided in Morning, Afternoon, Prime time, and Night, or could be viewed as a whole. This page has been removed in the current version because at the moment we only have one channel to display.

Media Feeds:

The "Media Feeds" feature was intended to provide the users with a facility to create and consume a constantly updated stream of clips based on the users' search criteria. It can be depicted as a kind of user specific stored search query which generates an up-to-date playlist every time it is visited. This feature was started, but abandoned shortly after because user studies showed that on the one hand, users did not understand the concept, and on the other hand found other features and their improvement more important.

Tell a Friend:

The "Tell a friend" feature which was located on the Clipdetail page allowed users to send a link to a clip to their friends by SMS or by email. This feature was

intended to improve the community building and help to spread the knowledge about our application. The feature was disabled temporarily because of the closed trial and to avoid content copyright problems.

Chapter 7

The M3 XP Process

The vast amount of published literature explaining different ways of doing Extreme Programming (XP) shows that in practice, there simply is no single approach. Even though XP is a simple and slim process whose practices are designed to be used together, it is usually tailored to the nature of each team and project in order to provide the maximum benefits.

Our team has been working on a project employing the XP methodology, experiencing unique issues arising from the distinct project setup and team composition, as well as the additional academic interests of the project. Initially, we aimed at applying "pure XP", but it became more and more obvious that for our project some of the practices just could not be applied in their "pure" form. Furthermore it was difficult to decide what the "pure" form actually is, because there are huge differences between the first and the second edition of "Extreme Programming Explained" [Bec99b] [BA04]. The first edition is relatively straight forward, although here also is mentioned that adapting and changing the methodology is allowed and maybe necessary. The second edition however, is more a list of possibilities than a concrete guide of how to do XP. Therefore, we tried to stick to the first edition and to implement the 12 core practices mentioned there. The concrete interpretation of these practices determines if XP can be applied successfully in the context of a team and a project.

In order to reach an optimized process for our project we continuously evaluated different approaches of applying XP practices and their combination, examining various aspects. The data was collected through code analysis, process evaluation, as well as notes of process retrospective review meetings and feedback from project partners. Furthermore, human aspects were taken into account by incorporating an experienced mediator into the team. The resulting metrics were analyzed, compared, and interpreted to find out the optimal set of practices. We have noticed that some practices can be adopted directly while others need to be tailored according to the unique environment. Here we outline our XP process and the optimizations.

7.1 Project Environment

We are developing a multimedia streaming application for mobile devices as a testbed to analyze the XP methodology. XP is being applied in a progressive manner: conscious of applying each practice, that can be applied, and looking for the improvement in the process at the same time. Hence, each team member is not only taking part in the software development process but is also making a critical

analysis of the way the process is being used. The objective of the project is twofold; to have a software product fulfilling the user requirements and at the same time optimization of the whole XP process as a profound academic research.

We are a team of six regular members: five developers and a customer and product manager. We have been working on the project since summer 2007. The project's duration was three years which was quite an appropriate time-frame for the development of the product as well as for the team members to conduct different studies and experiments regarding the XP methodology.

This project's main scientific and academic goal was the analysis of agile software development methodologies. Another goal was the research in the field of mobile application usability. Additionally, the business partners were interested in the commercial aspects of the project. [1]

7.1.1 Informative Workspace

A great deal of attention is necessary to provide an environment to support XP practices. Key factors in communication are the use of whiteboards, positioning and sharing of desk facilities to facilitate pair programming, stand–up meetings, developers buying-in to the concepts, the rules, and the practices of XP, and collective code ownership [GH01]. Communication and collaboration between customers, product managers, business partners, developers, and other stakeholders maximizes overall team efficiency [McN].

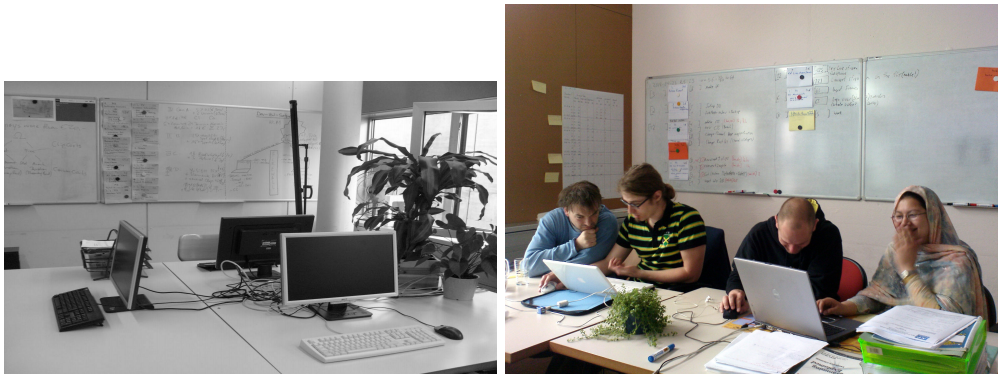


Figure 7.1: Pairing stations and individual places.

We sit side by side in a spacious room having enough space for private workplaces as well as for three separate pairing stations. This seating arrangement has promoted effective interaction in the team and has helped in resolving technical issues on the spot [LC05], besides it fulfills the human need for privacy. The face-to-face communication and instant help also have contributed in strengthening the relationships amongst the team members [LC05].

The use and placement of whiteboards is said to be an essential means of good communication XP teams [Bec99b]. The teams' XP room is equipped with six whiteboards which are used to record the XP stories agreed at release and iteration

[1] For a more detailed description of the team see Section 9.2.4, for the project see Section 4.

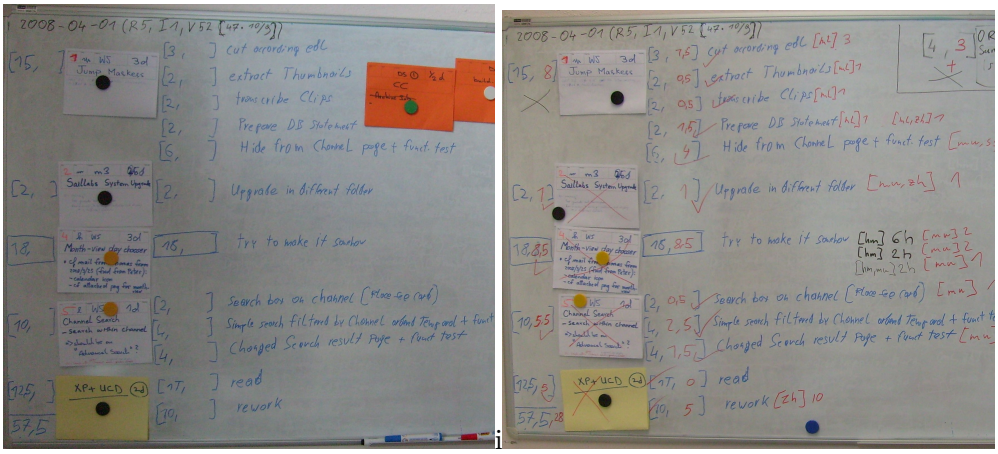
planning meetings. Story cards are physically stuck to the whiteboards in prioritized order with adjacent notes written on the board. Various graphs, showing architecture and velocity of the project, are also drawn on the whiteboards which therefore communicate the current status of the project.

Email, phone calls, and video conferencing are the tools used in routine communication with the usability engineers and with other partners. Personal visits to and by project partners are also made by the product manager and by other team members whenever necessary.

7.2 Collected Data

To review the development process, I collected empirical data describing our performance of applying the XP practices. This data is derived mainly from two sources: the project repository and the pictures of the Planning Game.

For the analysis of the project repository I used the build-in log function as well as the SvnStat [svn] tool. As we also checked in our documents and papers, I limited the analysis of our code repository to the application directory. From the analysis of the code repository it can be determined who has committed when and how many lines of code were changed. Additionally, we introduced a special format for the commit messages. We prefixed each message with the initials of the developer or pair that made the commit. Although this had to be done manually it worked remarkably well. After an initial phase almost all messages were tagged accordingly. Out of 1720 messages only 61 were not tagged and the majority of these messages was from the start of the project before the definition of the prefix. This additional tag makes it possible to investigate the pairing behavior of the developers during the project.



(a) Initial state (b) Final state

Figure 7.2: Planning and recording on the whiteboard.

The pictures of the Planning Game were taken for a period of about six months. Our whiteboard was used for planning and recording purposes during the iteration. At each Planning Game we took a picture of the end of the last iteration and the plan

for the new one as can be seen in Figure 7.2. These pictures allow distinguishing the types of stories, the estimate, and who worked on the story as well as the worked hours. The type of stories can be derived from the color of the story card. Application story cards are white, business cards are green, and science cards are yellow. The estimates of the stories and tasks are written beside the cards on the whiteboard. Written in red beside them are the developers who worked on a story and the hours they worked. Additionally, it can be seen if story is finished or not.

7.3 M3 Process Evolution

It was pre-decided that XP will be used as a methodology, therefore effort was made to establish a ground for its implementation. None of the team members had worked in an agile development environment before, so available literature, especially [Bec99b] [BA04], was used for initial guidance and reference. The team tried to apply all those main practices which could be applied at the initial stage of the project. Some of the basic practices were adopted fully, while others were implemented partially or in modified form.

7.3.1 Release Policy

We aimed to release a working version of our application to the project partners on a regular basis. In the early stages of the project, the duration of one release cycle was set to one month, enabling us to quickly get feedback on our work from the partners in order to sharpen our vision of the project goal. As the project took shape, the release size was gradually increased to two and finally to three months.

We were satisfied with a three-month release cycles, which complied with the quarterly planned business targets of the partners. But these releases were only internal releases for showing progress to our partners and therefore not taken seriously by all team members. The external releases we had in our project occurred not at these three month boundaries but in between. So we finally switched to an on demand release basis.

For tracking short-term progress, releases are further divided into iterations. Initially, we used a one-week iteration duration but later shifted to two weeks in order to reduce the administrative overhead added by the iteration planning meeting. However, the downside was that during the two week iteration many changes occurred which raised the necessity for re-planning. So we finally got back to one-week iterations which mostly could be held stable concerning the planned tasks. The aspect of planning overhead could be reduced by introducing a better and more efficient planning practice.

Current status: At the moment we do not have a fixed duration for a release because the project requirements have changed. We continue development and releases are made on demand basis depending on the business situation, that is, releases are scheduled when new business deadlines arise. The iteration length is now one week which allows us to keep the plan stable. This duration is now

preferred by all team members and was possible through an improved planning process.

7.3.2 Release Planning Game

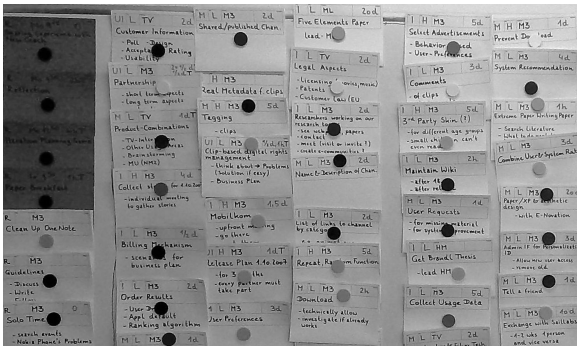


Figure 7.3: Selected story cards on the Release-Board.

In the beginning, our release planning meetings were attended by all project partners who, as stakeholders of the project, identified and defined user requirements, which were then formulated as XP User Stories [HLM+08a]. These stories were estimated by the developers with a granularity of days, prioritized, and selected for the upcoming release as long as they fitted into the available velocity. We also selected additional stories as backup if we would exceed the velocity.

It is to note that we used different velocities for releases and iterations as we found out that the granularity of story and estimate was different. Stories at release time were rarely concrete enough to actually implement them but sufficient for a rough estimate. During iteration planning time the details were clarified and the stories were re-estimated. It turned out that this estimates were almost never the same and sometimes differed greatly. Therefore, it became clear to us that this meant that we needed two different velocities. Additionally, we found out that the stories for an iteration rarely came from the release board. They were mostly rewritten or obsolete at iteration planning time. This was mainly because of the changing priorities and requirements during the release.

Later our release planning meetings were only internal as our project partners stopped to attend them. They were satisfied with expressing their wishes in meetings with our customer and project manager who then carried them into our Planning Game.

We finally stopped to make release planning meetings at all as we found out that they were not necessary for our project. They were substituted with small information meetings on demand where the general direction of the development was decided.

Current status: Currently we do not do release planning. We have small meetings instead where the latest business developments are communicated and the future development direction is decided. This direction determines the User Stories that

the customer brings up during the iteration Planning Game. During the release we simple implement as many stories as possible.

7.3.3 Story Cards

The stories generated during the release planning were written down on story cards and were prioritized by the participating partners. We distinguished between three main types of stories:

Application stories: These were the "traditional" XP story, representing features of the application.

Business stories: They described the diverse administrative and business activities, e.g., writing reports, collaboration with partners, meetings, presentations etc.

Science stories: They were only relevant for our team, hence they were not specified during the release planning meeting. They depicted the academic and research activities of the team, e.g., paper-writing, performing process analysis etc.

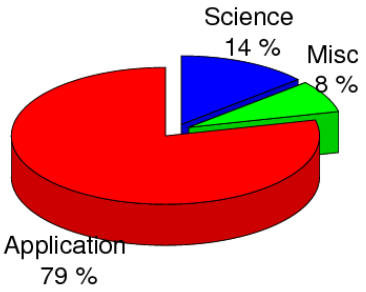


Figure 7.4: Time spent on different story types.

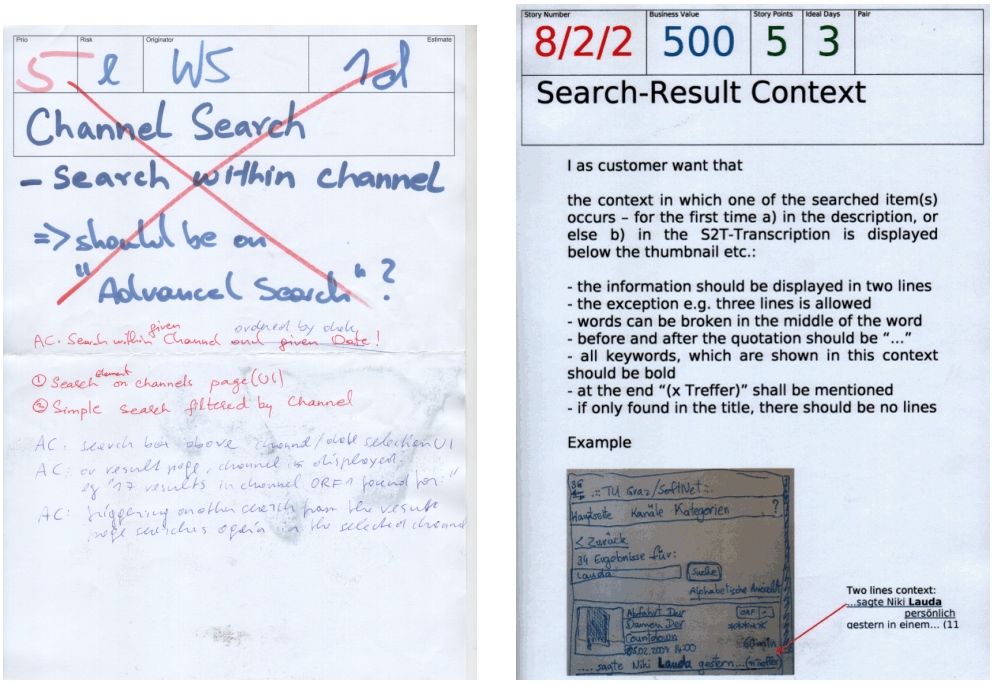
To visually represent the different story types, we used a simple color encoding: Application story cards were white, business cards were green, and science cards were yellow. Figure 7.4 shows the distribution of the story types in our process.

Later we decided to plan only programming stories as we had some difficulties with the other approach. The major problem was that the science stories, e.g., paper writing turned out to be not possible to plan at all. Therefore, they were excluded from the velocity and were done by the developers on their own. The reason for the removal of the business stories was that they mainly concerned the customer and rarely developers were involved, but the customer's work was not part of our velocity.

Current status: We created a new format for story cards which can be seen in Figure 7.5b. The narrative of the story card is now checked against the "5 Ws":

Who: Indicates the type of user.

What: Tells about the action the user wants to perform or the request he has.



(a) Old format. (b) New format.

Figure 7.5: Example of story card formats.

When: Describe when this action should be performed.

Where: Where on the page or in the application.

Why: What is the motivation.

Additionally, the card contains scenarios or mock-ups which are also used as acceptance criteria.

Example Story:

```
Content Categories

I as a User want… (Who)
to have the content divided into different categories (what)
so that I can see all existing content according to its genre. (why)
(Categories instead of media feeds)

Scenario:
User A clicks on "Categories" on the main menu bar (where)
He comes to a content page where he can select the content
with a dropdown box (when (after clicking) + where)
He selects a category and on the page he sees the
clips ordered alphabetically by title
```

Also the header of the card has changed. Beside the story title we currently have a fields for a unique story number (e.g., 8/2/2 = Release 2/Iteration 2/Story 2), Business value (0-1000), story points, the signing pair, and ideal days. The ideal

days field is a current experiment to find out if there is a correlation to the story points.

In Figure 7.5 it can be seen that the new story card format is an improvement. Apart from the improved visual appearance, the story description in the new narrative form is now much more concrete. The inclusion of mock-ups and scenarios contributes to the clarity of the story.

7.3.4 Iteration Planning

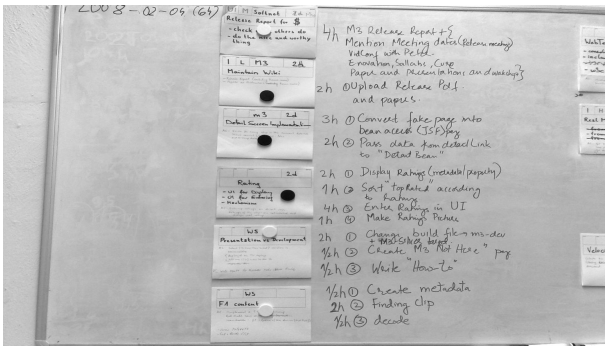


Figure 7.6: Selected story cards on the Iteration-Board. .

The iteration planning meeting was held at the beginning of each iteration and was attended only by the team members including the product manager. The product manager took the role of the on-site customer. He selected and prioritized stories which were planned for the current release for the upcoming iteration. The stories were explained and discussed. Then they were broken down into detailed tasks and were estimated by the developers. The intended results of the tasks were explicitly defined by writing acceptance criteria. The number of stories which fitted into one iteration depended on their estimation and the available velocity. Before and after implementing these stories, continuous feedback was obtained from the usability engineers and the stories were modified according to their feedback.

The main bottleneck of this approach to planning was the long lasting discussions. The customer brought up the stories during the Planning Game. They were written from his point of view and usually needed a lot of clarifications from a developer's perspective. E.g., stories like: "I want to be able to search in the clips." are too vague for developers. Here crucial information is missing, like: How to you enter a query? Where are those elements located? How should the result look like?

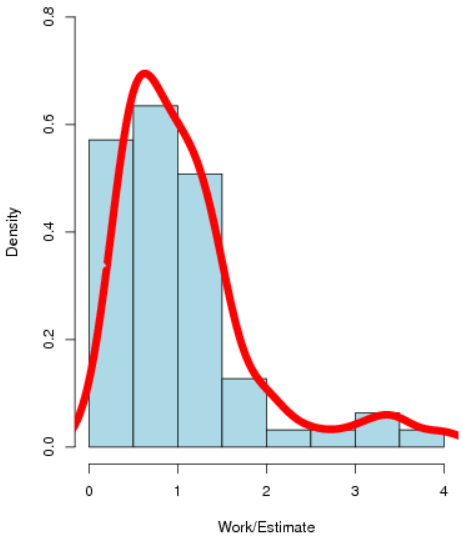
Asking those questions during planning time usually lead to confusion and uncertainty on the customer's side and to a discussion amongst the developers about the various possibilities. Once the story clarification was finally finished, the next discussion began among the developers about the task breakdown, how to implement it, and about the estimation. Altogether we usually needed about 3-6 hours for the planning of a one-week iteration, which was far too long.

Later we improved the Planning Game significantly. The stories were now prepared by the customer together with one developer before the Planning Game. The

Planning Game itself was further optimized by using planning-poker cards for estimation, skipping the task breakdown of the stories, and introducing stories of smaller scope instead. Through this process we were able to reduce the planning time significantly to about 1-2 hours, which allowed us to go back to one-week iterations.

Iteration	Est.	Worked	Ratio
R3I3	13	20.0	1.54
R3I4	4	4.0	1.00
R4I1	4	4.0	1.00
R4I2	11	14.2	1.30
R4I3	40	33.2	0.83
R4I4	42	60.0	1.41
R4I5	5	4.5	0.90
R4I6	44	53.5	1.22
R5I1	42	25.0	0.60
R5I2	46	31.8	0.70
R5I3	26	30.2	1.19
R5I4	12	10.5	0.91
R5I5	5	3.0	0.60
R5I6	26	35.8	1.38
Mean	22.79	23.55	1.04

(a) Worked to Estimated hours per Iteration



(b) Histogram and density of the estimation ratio distribution.

Table 7.1: Worked to Estimated hours per iteration and density distribution of this ratio from the individual stories.

For some time we tracked the story estimate as well as the actually worked hours. This allowed us to investigate the distribution of the estimation error in our process which can be seen in Table 7.1a and Figure 7.1b. For this distribution 63 stories of 14 iterations were examined and the ratio between the worked and the estimated hours was calculated. The average ratio surprisingly turned out to be 1.04 and the distribution shows that the majority lies below two which correspond to the known rule of thumb in SW development. It is also interesting to note that in general there is a slight tendency to overestimation.

During planning we additionally planned a few more stories and reserved them in case we would exceed our velocity. But most of the time, additional stories were not planned at all. Figure 7.7 shows that only 2% came from our reserved stories and over 14% of all work was done on stories which were not planned. This occurred mostly because of new developments from the customer side and was one of the main reasons why we switched back to one-week iterations.

Current status: Our current iteration planning process is as follows: The customer together with one developer prepares his stories before the Planning Game. In this way, the story is already clarified from a developer's point of view, which allows it

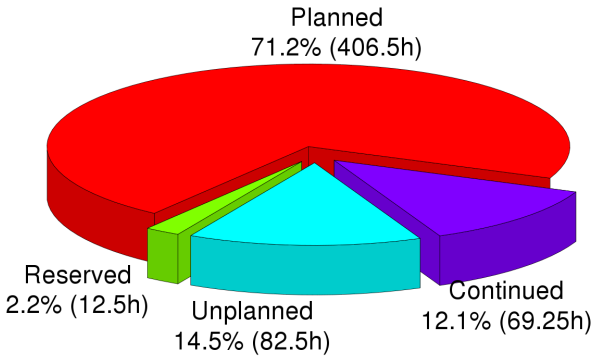


Figure 7.7: Distribution of the work per story planning status.

to be estimated. Additionally, the developer can advise the customer and tell him what is possible and how complicated which solution will probably be.

As we have now one-week iterations, the stories are of a scope of one to three days with a preference for smaller stories, making the task-breakdown obsolete. Here again the supporting developer has an important role in guiding the customer because he has a better feeling about the size as well as if and how the story has to be split.

During the Planning Game we use planning-poker cards to estimate the story in story-points. The story is presented and explained by the customer and the supporting developer. The other developers ask questions if something is unclear. It is to note that there usually are very few questions, because of the pre-clarification during story writing. Afterwards each developer selects a poker-card from his set which represents his estimate. At a signal from the customer, the cards are shown simultaneously which prevents mutual interaction.

For most of the story-cards, the estimates are more or less equal and an approximate average is used as final estimate. If the difference of the individual estimates is greater the reason is discussed, as this indicates that there are probably misunderstandings or very different implementation ideas. After this discussion the estimation process is repeated.

If the estimates are altogether very high this is an indication that the story has to be split. This is done immediately and the resulting stories are estimated as described above.

In this process, the developer which supports the customer in the story writing process changes regularly so that each one has this experience. The close interaction of customer and developers slowly brought a mutual understanding, as the customer realized what information the developers need and vice versa. This understanding had the additional benefit that the new stories from the customer are formulated in a much more concrete way which eases the story writing process.

7.3.5 Pair Programming

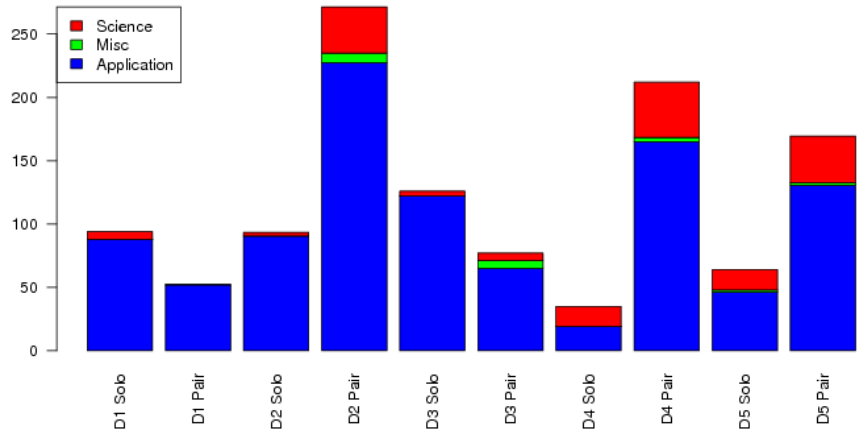


Figure 7.8: Hours worked in pair and solo per developer for different story types.

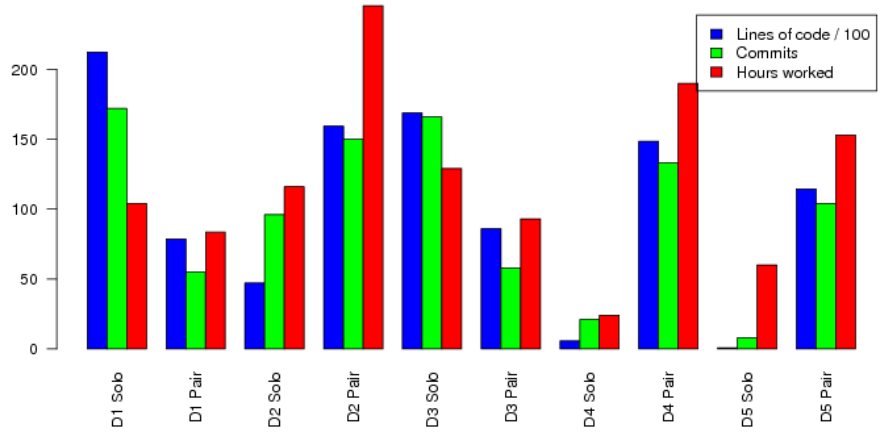


Figure 7.9: Pairing behavior of the individual developers concerning lines changed, commits, and hours worked.

Pair programming means that all program code is written by two developers working together on the same machine with one screen, one mouse, and one keyboard [Bec99b] [BA04]. This practice helped us in spreading and sharing the project-specific knowledge and improving the technical skills of the developers.

In the beginning we tried to strictly stick to this practice and really do everything in pairs. We also applied working in pairs to non-technical stories. For example, the product manager paired with a developer for writing customer-acceptance tests and business assignments requiring technical knowledge. This way the customer

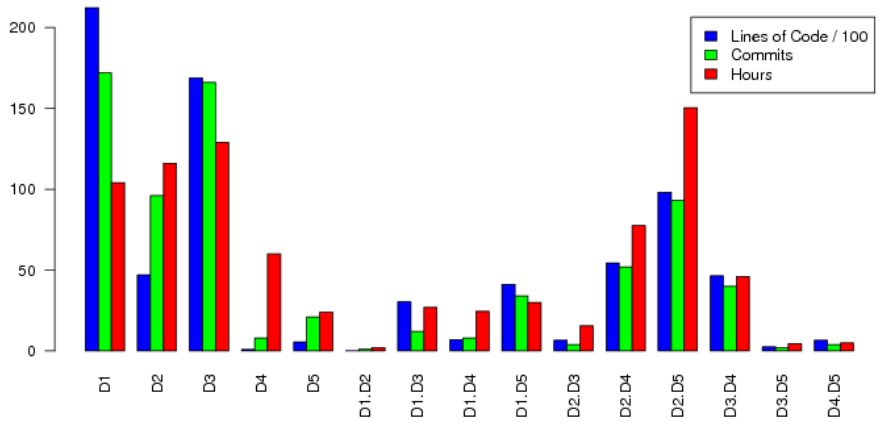
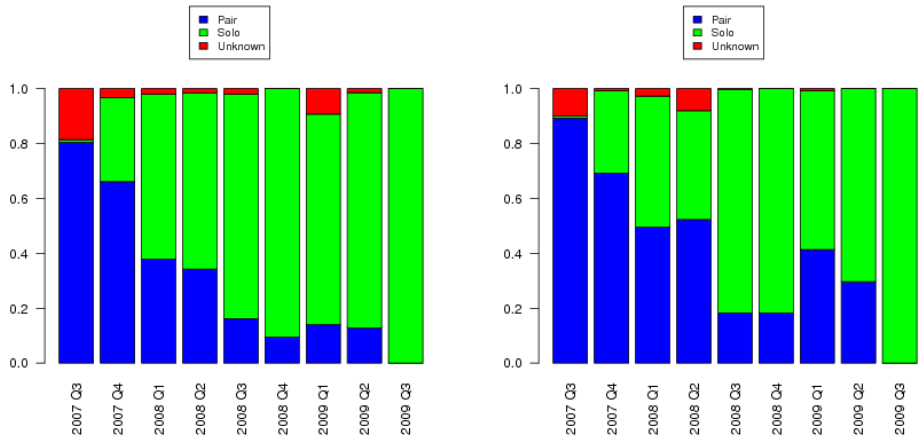


Figure 7.10: Pairing behavior and favorite pairs concerning lines changed, commits, and hours worked.

learned about the process and the internal status of the application which helped in better understanding and implementing his requirements. We also worked in pairs for research stories. However, we had troubles with this approach.



(a) Commits per quarter.

(b) Line of code changes per quarter.

Figure 7.11: Pairing development through the project.

Later we used the pairing only for programming task as there was some opposition to use it for other tasks. But unfortunately also the programming tasks were often done solo or partly solo. This was due to different habits of team members, tensions in our team, different skill levels of team members, and opposition against the strict pair programming paradigm. Developer D1, D2, and D3 are from Austria and experienced developers while D4 and D5 are scholars from Pakistan. So they have

a different cultural and educational background which affects their working and pairing behavior. Overall we recorded 412 solo and 782 pair hours from which the application stories used 365 respectively 640 hours. This means that we had almost twice as many pair hours in the recording period than solo hours. However, for an XP process this amount of solo work on the application is very high.

The distribution of work between our story categories is shown in Figure 7.8. It is obvious that the amount of science work is the highest for D4 and D5 regarding solo as well as pair work. D2 has an equally high rate but only when pairing. The educational background explains why D4 and D5 did much more science work and less work on the application.

Figure 7.9 shows the differences between the individual developers in their attitude to pair programming. According to the data, D2, D4, and D5 are mainly pairing while D3 and D1 are mainly working solo. Furthermore it can be seen that D4 as well as D5 are almost doing no solo work. Especially not on the application as the low number of code changes indicates. But in comparison to their solo work, they have a high pairing activity.

The pairing behavior of the developers can be seen in Figure 7.10. It indicates that there are certain preferences in pairing. According to the data, the pair D1.D2 has the lowest pair ranking while the pair D2.D5 has the highest. Furthermore it is to notice that the pairs with the highest activities always contain a member from Pakistan and one from Austria. Pairing among Pakistani developers as well as pairing among Austrian developers was not very popular.

Figure 7.11 shows that the amount of pairing during our project was steadily decreasing. In the beginning we had almost exclusively pair work, but soon after, the solo work increased until there was rarely pairing. It can be seen that we also had a few unknown commits. This was due to the fact that especially at the beginning we were not use to use the tagging and the format was still under development. Additionally, during branching and merging, because of the used tools, the tagging information was often forgotten.

The statistics derived form the code has several drawbacks. High changes in changed lines of code do not necessarily indicate a high development activity. Some of our changes were caused by reformatting code, either manually or with the IDE auto-formatter, a task which was especially carried out often by D1 and D3 and which usually affected many lines at once. Also the numbers of commits give no clear indication as the commit behavior and frequency varies among developers. However, all together gives a pretty good overview about what happened in the team. This picture is confirmed by the manual recording of our working time. Although it was not recorded during the entire period, they show a trend which fits well to the data derived from the code. This data also confirmed my subjective personal perception as part of the team. An additional analysis of the pairing situation is given in 9.4.4.

Current status: Currently pairing is desired but not mandatory. It is up to the developer which is responsible for a story if he pairs. This developer usually asks others to pair, but if none is available he can also work solo. As this solution has removed resistance against pairing and much tension in the team it is far from

optimal.

7.3.6 Collective Code Ownership, Refactoring, and Coding Standard:

A Subversion repository is used for managing the single code base and the code is shared by all developers. Whenever a chance for code improvement was identified and there was enough time at hand, the required actions could be performed on the spot. The changes were communicated in the stand-up meetings, during pair programming, and sometimes through a short ad-hoc discussion involving all developers.

To allow this we introduced a common coding standard. It took a lot of discussion until we finally had this coding standard as the developer had different experiences and therefore different opinions on how it should look like. In the end we agreed to a very open one.

However, the standard was so open that the coding styles of the individual developers still differed greatly. This resulted sometimes in refactorings where mostly the coding style was changed so that the acting developer felt more comfortable in reading the code. Which of course made the original author unhappy and she sometimes changed that part of the code back to its original layout.

Major refactorings were rarely made as it was the opinion of the team that these should be scheduled as separate stories. But for the customer because of time constraints they were usually less import than features. Additionally, some developers were new to the concept of refactoring and sometimes the necessary test support was missing. Refactoring of code was not a routine practice in our team but was done on an on demand basis. Meaning, whenever a developer saw the urgent need to improve the code or when we needed to substantially extend the application functionality and had to prepare as sound base.

Current status: The collective code ownership works now as the coding standards converged over time. Refarctorings are now made more easily as we decided that we do not plan refactorings separately but treat them usually as parts of the stories.

We also came to the understanding that in an XP project not everything hast to be done at the spot. That means that transitions of technology can be done partially and don't have to be done in a single big effort. Thus the refactorings can be spread over multiple stories, making it even safer as just a bit of the application is changed at a time allowing quick corrections. We have a rule of thumb that says that refactorings should be made up to a point where they do not take longer as the story itself.

Refactoring has become a routine task which is usually executed regularly during development. Before the implementation of a new story is started the existing code is refactored if necessary, during the story refactorings are made to keep the code simple, and after the implementation is finished, refactorings are made to clean up and leave a consistent piece of work.

7.3.7 On-site Customer

As the target group of the product being developed by our project is manifold, we cannot directly implement the on-site customer practice. Therefore, our customer is in reality a so called customer proxy. Initially the product manager as well as the developers played the role of the customer. Soon many shortcomings of this approach became apparent. Each one had an own opinion and the absence of common acceptance criteria for the stories resulted in long discussion cycles during the planning meetings as well as during implementation. We also felt difficulties in the prioritization of the stories. To overcome those problems, we decided that only the product manager who communicates with our project partners should play the role of the customer.

This worked well, except sometimes when the project manager, which also acted as customer, and the product manager were not of the same opinion which could lead to contradicting information for the developers. During the later phase of our project the two got more experienced in clarifying story details beforehand, but nevertheless misunderstandings could not be prevented fully.

Current status: At the moment only our product manager acts as on-site customer and therefore single point of contact for the developers. He is in constant contact with the product manager and the Usability Engineer (UE) to clarify stories details. Additionally, a mutual understanding has developed between the developers and the customer. The customer now understands what information developers need and developers know now what the customer wants. This, in conjunction with the improved planning process has led to fewer discussions, clearer and more concrete stories, and a similar vision.

7.3.8 Metaphor

As everyone was new to the process and to the project, there was no common understanding of the metaphor – the shared terminology about the project and the process [Bec99b] [BA04]. This resulted in misunderstandings about the features to be implemented which eventually led to the delaying of their delivery. It also expressed itself through the different terminologies used in naming variables and functions in the code.

The release planning meetings with our partners, our internal iteration planning meetings, stand-up meetings, retrospective meetings, as well as pair programming have contributed to evolve our metaphor and it is still evolving. But this is happening unconsciously as we do not pay much attention to it. There was no attempt to actively create a common metaphor.

Current status: Currently we have a shared understanding of the application which we call metaphor. However, it is an unconscious metaphor as we did not explicitly formulate it nor made attempts to create one. The evolutionary and subconscious character of our metaphor can be seen when new features are introduced. As no effort is made to consciously find a metaphor, names are chosen arbitrarily at first

and it takes some time till they converge.

Our so called metaphor works well on the system level where the names of variables and functions have helped to spread a common vocabulary through the team. However, I doubt that if asked for a short, one sentence description of our project, the team members will give a common answer. According to Beck the metaphor should "Guide all development with a simple shared story of how the whole system works." [Bec99b]. Therefore, I think this practice is not really implemented, although we may have unconsciously similar thoughts. It needs improvement as it represents a common vision and goal which is currently missing or could not well be communicated so far. But it would be worthwhile to develop as it could provide clarity in the decision process and focus during development.

7.3.9 Simple Design

Simplicity is described as "the art of maximizing the amount of work not done" [Man01]. From the very beginning the idea was to keep the process and the design as simple as possible. We started with hand-drawn paper prototypes to visualize the customer requirements and to show the customer how the requirements would be realized. Refactoring of code also should have contributed in keeping the design simple.

However, due to lack of experience with this simplicity principle the design tended to get to complicated. Possible future extensions were anticipated, large scale frameworks were introduced before they were needed, and rare possible cases were discussed longer than necessary before the main functionality was implemented and they could be tested.

As the simple design principle contradicted with prior experiences and also with the nature of some of our team members, there were long and fruitless discussions about this topic. Arguments about best practices and state of the art (e.g., "A web application has to have JSF, Spring, Hibernate and has to have three layers") were used constantly, forgetting that "state of the art" not necessarily meant "state of the XP art". Also some developers, tired of these discussions, refactored the application and introduced these framworks themselves. Which ultimately lead to the current application and, in my opinion, made the consequent development much more complicated than necessary.

Later in our project these problems occurred less often, although the principle understanding and willingness to do "the simplest thing that could possible work" did not increase much. But as the application got more stable, the principle – and mostly not simple – design decisions had already been taken. Therefore, only decisions of smaller scope were necessary which had less potential for discussions.

Current status: Simple design is no longer an issue as no real design changes occur at the moment. This does no mean that we now follow this practice. We had some small improvements over time, but all in all this practice was and is not properly implemented. Probably this can partly be attributed to the fact that we are all part of the scientific community. In science, exact data and throughout investigation, even of rare cases, is mandatory and rough, simplified solutions are not desired.

This educational background may have contributed to the inability to apply the simplicity principle.

7.3.10 Test-first Programming

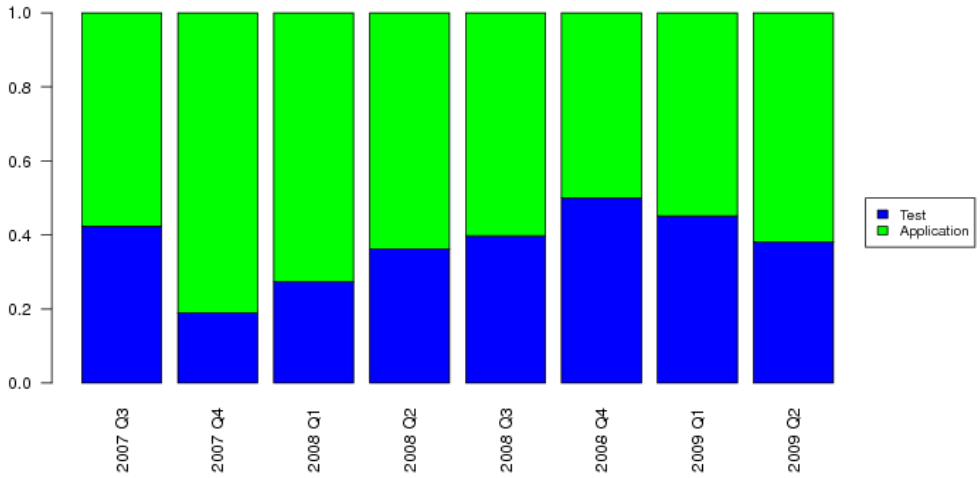


Figure 7.12: Ratio between application and test code per quarter.

As all team members were new to XP, it was difficult to follow the XP practice of Test-driven Development (TDD). Writing failing automated test before any code [Bec99b] [BA04] was contrary to the development style most of the team members had used for years. Especially during the first releases this was a problem. For the subsequent releases, the implementation of this practice has improved. Figure 7.12 shows the amount of lines changed in the test code compared to the application code. It can be seen that there is a steady increase over time as the team got accustomed to TDD.

We also introduced automated acceptance tests which were written together with the customer to get clear acceptance criteria and to ensure that the requirements were met. However, it turned out that this was even more difficult than writing the unit tests. The problems were partly in the different understandings, in the availability and willingness of the customer for these tasks, in the missing experience of the developers which worked together with the customer, and in the limitation and complexity of the testing framework used.

It turned out that, as we were developing a web application, these tests were difficult to run locally. Also we found that in including them into the continuous integration process we produced only failing builds until the end of the iteration which was demotivating and also dangerous as it could have hidden real errors. Therefore, we skipped the automatic acceptance testing and tried instead to formulate concrete acceptance criteria for stories.

Current status: Through practice and experience the ability and willingness to do test-first programming has increased. However, depending on the personality there is still some reluctance. This is especially visible if time pressure is present.

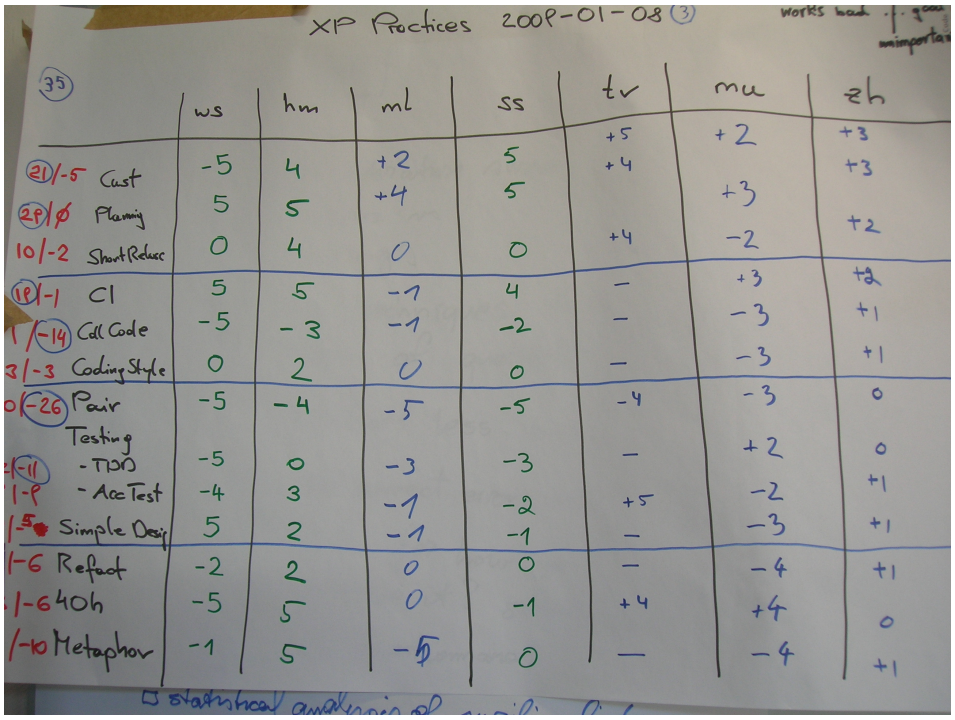


Figure 7.13: Reflection chart showing the ratings for the different XP practices. Measured was the satisfaction of each individual team members with the adoption of these practices.

7.4 Conclusion

After working with XP practices for almost three years, our experiences show that most of the practices are helpful for a project with multiple objectives (in our case, research, application development and business). Pair Programming (PP) helps in spreading knowledge, Simplicity saves work, TDD and Refactoring provide clean code and a tested system, etc. Additionally, the benefits of co-location, face-to-face communication, stand-up and planning meetings, as well as retrospective review meetings have contributed to improve not only our process but also to increase the overall morale of the team.

However, the low ratings of some practices indicate that our team still needs more experience to properly apply them. This was also confirmed by results of our reflection meetings as can be seen in 7.13. We experienced problems especially in TDD and PP. These are practices which are rarely thought or used and therefore contradict previous experiences of developers. Additionally, PP forces the team members to have a much more intimate contact than usual. The necessary unlearning of old behavior and the close interaction makes these practices harder to implement than practices like Continuous Integration (CI) or Collective Code Ownership.

Chapter 8

M3 Usability Inclusion

We are developing an application that enables a user to perform content-based search for AV material and play it on a mobile phone. This content includes radio and TV archive material, like documentaries, discussion programs, movies, music videos, music, etc. The major problem for an average user in this context is the combination of the overwhelming amount of multimedia content available and unsatisfactory User Interfaces (UIs) for accessing it. Usability is the key success factor for such applications.

User-interface design determines the success or failure of almost any application. AV consumption on mobile phones will be accepted only if users can easily find what they are searching for. But search on a mobile phone presents unique challenges as compared to search on a PC [McL07]. Appropriate UIs are therefore crucial for enhancing the user experience and enabling the effective consumption of mobile content [SY06]. Therefore, our focus is placed on the combination of iterative UI design and user-centered application design.

Being integrated into agile methods, User Centered Design (UCD)/usability engineering helps to reduce the risk of running into wrong design decisions by involving real users and results in increasing the acceptance of software applications [MRH07]. We have integrated Extreme Programming (XP) and UCD in our project regarding a multimedia streaming application for mobile phones since summer 2007.

There already exist approaches of combining agile methodologies and Usability Engineering ([HES+05, MM05, CL03]). However, to the best of our knowledge none applies our specific composition of practices. We combine XP [BA04], UCD, and an iterative user-interface design approach utilizing different Human Computer Interaction (HCI) instruments.

Although we do not involved the end-users directly, the HCI instruments allow us to involve them indirectly in the process by user studies, Extreme Personas (a variation of the Personas approach), usability expert evaluations, usability tests, extended unit tests (automated usability evaluations), and lightweight prototypes [WTS+08][HLM+08c].

8.1 Our Process

Projects involving XP need quick feedback and ad-hoc input during development. For this purpose the traditional usability methods have to be tweaked in a way to be less time and cost intensive. The usability method we used most was the expert-

based usability evaluation. To get quicker feedback we involved fewer experts than recommended (mostly only our Usability Engineer (UE)) and usually evaluated mainly single stories instead of the whole application. As our UE was not on site, we got his feedback by IM (Instant Messaging), email, or (video) conferencing after sending him the mock-ups of our current stories [WTS+08][HMS+09].

The following sections describe the process which was followed for the user-centered design of our application.

8.1.1 Approach to User-Centered Design

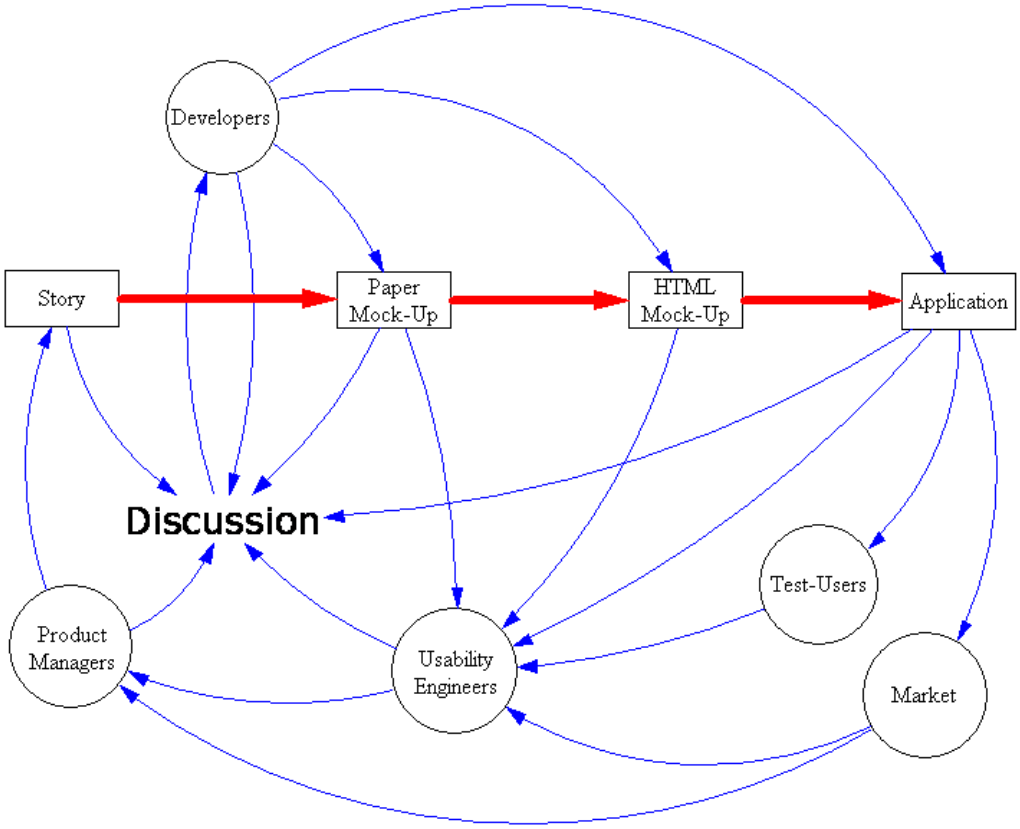


Figure 8.1: Iterative UI design workflow.

User-interface design plays an important role in the acceptance of a web based application by users. The overall process of our approach to UCD is based on evaluating the usability of the application in small iterative steps. This helps us to gain insights into the functional and cognitive requirements of real users. Additionally, we evaluate also the User Stories before the actual planning and implementation to avoid usability problems instead of fixing them. We design prototypes of the user-interface of the system and test them throughout the development process. As a result the fidelity of these prototypes increases and evolves.

The workflow presented in Figure 8.1 illustrates the iterative design approach to incorporate UCD into our XP process. From a broad perspective, the application development cycle starts from defining the User Stories (user-required application

features), then comes to mock-up designing and at the end the actual implementation is performed.

In the beginning of our project the process was executed as follows:

- Different feature-related User Stories of the application were created by the customer in coordination with all the stakeholders.
- Developers created different paper mock-ups for each of the required features to collect ideas and to present them to the customer.
- The customer decided which of the mock-ups best suited his needs, or he suggested modifications of the mock-ups.
- A final mock-up was derived according to the customers' wishes which then served as the basis for the actual implementation.
- Once an implementation mock-up of a feature (or a group of related features) was finished, the usability engineers were asked to give feedback on it.
- After incorporating the feedback given by the usability engineers into the application, end-user tests were conducted on the application by the usability engineering team.
- The feedback on the application from the usability engineers, as well as, from the test-users was taken as input for further refinements of the UI design of the application.
- The results were incorporated into automated tests, by employing test-driven development, which were used as an executable specification for the actual implementation.

This feedback-and-change cycle provides insights into whether the user-interface design is meeting the usability criteria. As the application development is done in short iterations, the developers are able to refactor the system continuously according to the feedback derived from the parallel, as well as iterative, UI design process. Hence, the system evolves according to the needs of the end-users and the specifications derived from actual usage.

During the course of out project we optimized this approach. The current workflow is depicted in Figure 8.2 and consists of the following steps:

- When the customer introduces new features, he writes them together with a developer in the form of XP User Stories. Already during this process mock-ups are drawn and the UE is called for advice, which shortens the first feedback cycle. The resulting story is already usability tested, and the final paper mock-up is part of the story card.
- If necessary, developers create a dummy implementation in the system as high fidelity mock-up for further clarification. Because the expert evaluation of the paper mock-up had already happened, this step was rarely necessary, and if it was done it was more accurate and could be reused.
- The developers implement the story. During this phase the UE and the customer are asked for clarification if problems arise.
- The UE analyzes the finished stories at the end of an iteration. Also a usability tests can be conduced if the status of the application permits them and they

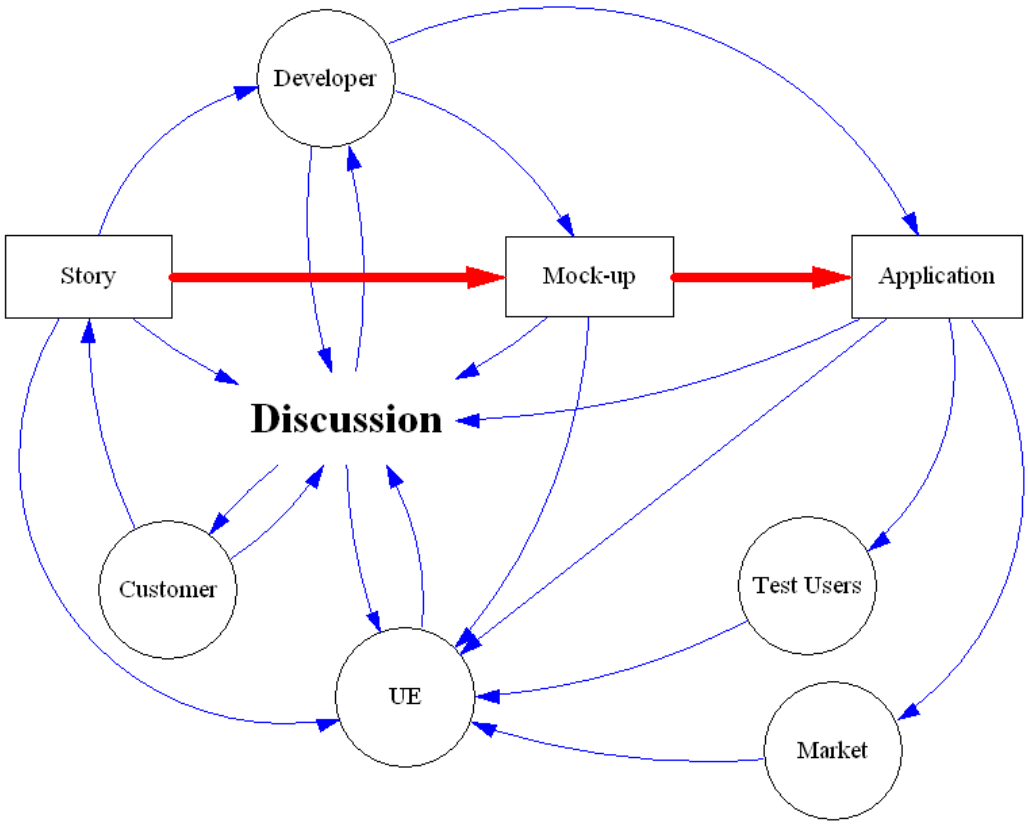


Figure 8.2: Final iterative UI design workflow.

are affordable. The feedback on the application from the usability engineers, as well as, from the test-users is taken as an input for further refinements in the UI design of the application into the next iteration.

- The results are incorporated into automated tests, by employing test-driven development, which are used as an executable specification for the actual implementation.

Although this approach uses almost the same layout as the previous one, it is more efficient. We have shorter feedback loops and more ad-hoc input instead of the more formalized process described above. The feedback is more fine-grained as it only is concerned with the currently implemented stories and not so much with the whole application, which makes it more usable and more likely to be incorporated.

Also the central part, the discussion, is minimized by involving less people. In the first process the User Stories were clarified during planning with the whole team and after that again discussed with the UE. In the later approach it is clarified prior to the planning among three people (customer, one developer and the UE) and is at planning time already concrete and evaluated.

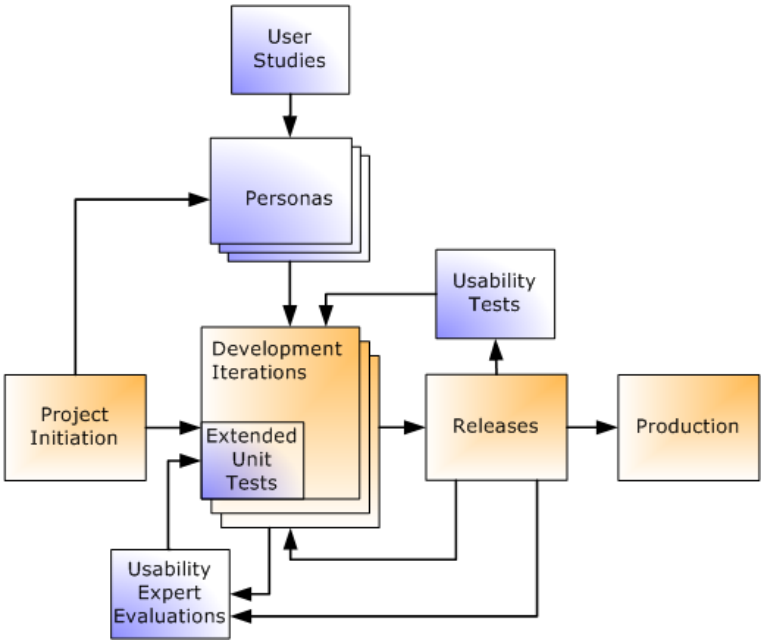


Figure 8.3: The integration of HCI instruments into XP [WTS+08].

8.1.2 Integration of HCI Instruments

Figure 8.3 describes the way the different HCI instruments (User studies, Personas, extended unit tests, usability tests, and usability expert evaluations) are integrated into our XP process [WTS+08]. When applied correctly in various phases of the project, the instruments are designed to reach the goal of improved software quality not only in terms of technical quality but also in terms of usability [WTS+08].

End-users are integrated in two different ways. On one side, user studies are performed and used as an indirect end-user input for the development process. They are taken into account to develop Personas [JF05] which then specify the direction of development by guiding the customer while identifying and creating User Stories. At the end of a development iteration the user studies broaden the vision about the users which allows to extended and adapt the Personas. On the other side, feedback from usability tests performed by test-users as part of the usability evaluations serves as a direct input for further enhancement and development of the application [WTS+08].

8.1.3 Usage and Development of Mock-ups

As stated above, we make use of two different types of mock-ups; low fidelity paper mock-ups and high fidelity implementation mock-ups. In our case usually a paper mock-up is sufficient and only in some cases an implementation mock-up is needed. As we have the benefit of an on-site customer co-located with the development team, for these tasks a quick implementation mock-up is designed and immediately shown to the customer. That implementation mock-up is then modified based on the immediate feedback from the customer.

The approach combines the quick feedback-and-change cycle of hand-drawn paper mock-ups with the more time-consuming process of computer-based prototypes. Paper mock-ups are used to get the basic concepts right while software mock-ups are used for a more detailed view.

In the first stages of our project we created intermediate HTML mock-ups, but later we omitted them and used dummy implementations inside our application instead, meaning we included the page without functionality. This type of high fidelity mock-up had the advantage that we could already see during the implementation of the mock-up if the desired design could be done with the used web technology. Additionally, development effort was saved because the final version could be reused by filling it with functionality.

8.1.4 Frequency of End-User Tests

For running usability tests with test-users three preconditions should be met. First, from the business point of view it should be an appropriate time. This means that the system should stay more or less stable and only small changes and usability enhancements are planned in the near future. Otherwise the results of the tests are obsolete before they are delivered.

Second, there should have been enough new functionality added to the application that it becomes effective to perform usability tests before continuing with further development. In this way, the expensive part of involving real users can be done more effectively.

Third, it is important to choose representative test-users from different age groups, bearing in mind the targeted customers for the proposed service.

Unfortunately real end-user tests were just done once in our project and this was too early and during a period of major changes. It would have been better if user tests could have been made on a regular basis, like at the end of each release, but considering the expenses and resources required this was not possible.

8.1.5 Testing Issues

A big issue in mobile user-interface design practice is that current approaches are not sufficient for mobile phones [SY06]. Paper prototypes or screen shots are good and sufficient for verifying non mobile-based product requirements, but in case of applications for mobile phones they are not sufficient for finding out and solving usability issues related to detailed interaction on the small device with its limited user input capabilities [KK05].

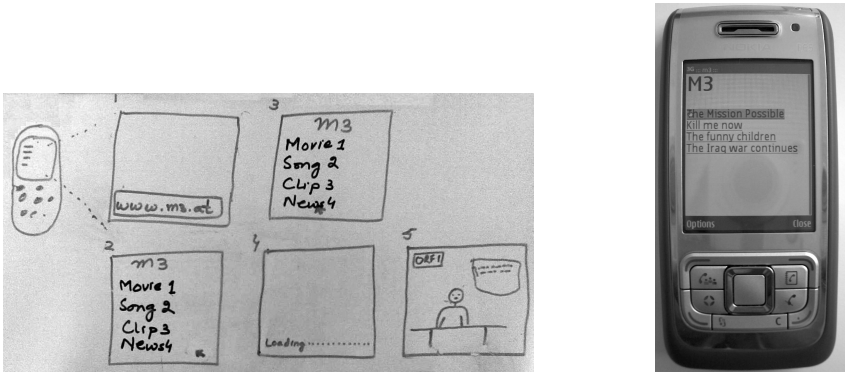
Therefore, in our case the application is tested on mobile phones and not on any web based simulator for understanding the interaction issues concerning the use of mobile phone interfaces [KK05].

Another issue was brought up by the idea of automated usability evaluations and extended unit tests. Unfortunately to implement usability related unit tests is very cumbersome as they are not readily supporting by common testing frameworks. Also the inclusion of existing tools like the W3C Validator [wwwi] is not easy

because they lack defined interfaces.

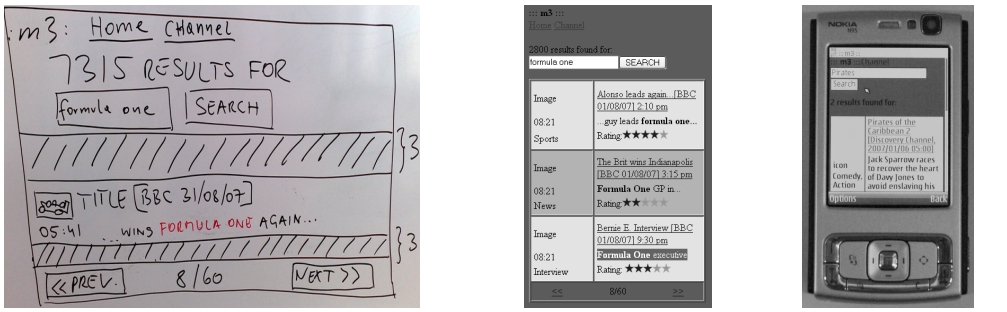
Because the development of more general tests was not supported, we tested individual findings without the possibility to generalize them. E.g., it was not possible to check on each page that every button label starts with an uppercase character, so we had to test each button separately. This made the test very vulnerable to changes in the layout, but nevertheless proved to be helpful as it detected each small change.

8.2 Process Examples



(a) Paper mock-up. (b) Application on mobile.

Figure 8.4: From paper mock-up to mobile: the first search-results screen.



(a) Paper mock-up. (b) HTML mock-up. (c) Final Application.

Figure 8.5: An additional HTML mock-up: a refactored search-results screen.

As the application development is done in short iterations, the developers are able to adapt the system continuously according to the feedback given. Hence, the system evolves according to the needs of the end-user and the specifications derived from actual usage. Here I show examples how this process was actually carried out in our project.

For the paper mock-up in Figure 8.4a the usability engineer raised the following issues:

- Missing strategy for displaying larger result sets (balance between pagination and scrolling).
- Missing feedback mechanism to highlight the pointed-to item (especially needed in unfavorable lighting conditions).
- Undefined application behavior after playback of the clip ends (e.g., no return option specified).

Figure 8.5 shows the mock-ups of an improved version of the same feature. In addition to the process depicted in Figure 8.4, in this scenario an HTML mock-up was created after the paper mock-up. The design was derived from the following user scenario (a so-called User Story in XP [BA04]):

Search results presented to the user should contain clip-related information which can aid the user in choosing the clip. Also, the context in which the keyword was found as well as the number of search results should be visible. Furthermore it should be possible to start a new search immediately.

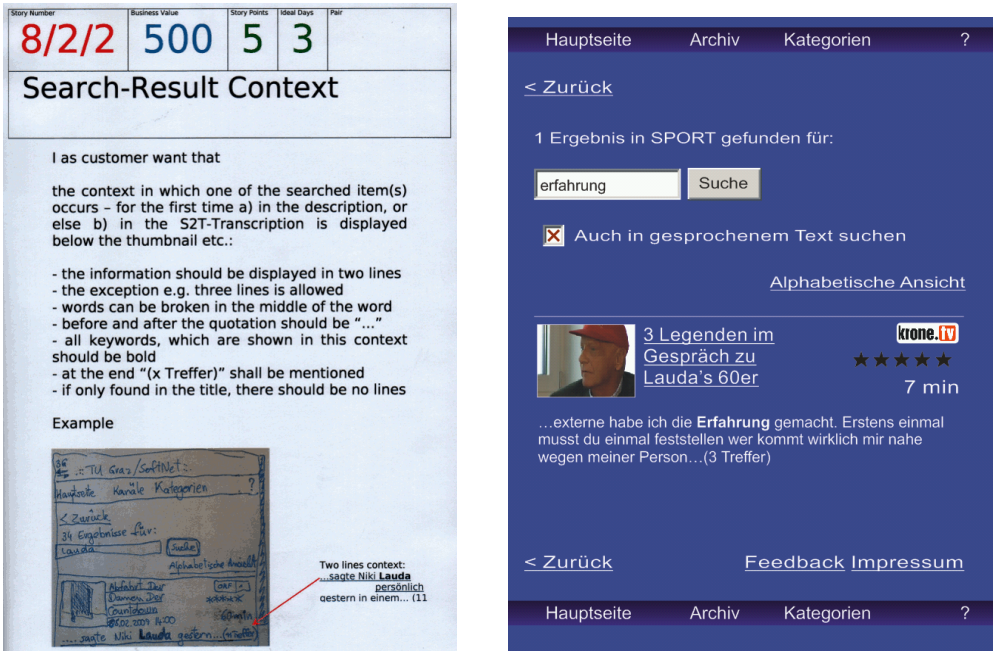
It can be seen that two issues from the previous feedback, namely pagination/scrolling and item highlighting, have been addressed in the refactored design. On this refactored design presented in Figure 8.5, the usability engineer provided the following feedback:

- Forms: It is common to leave some white space between text-input-field and the button. For GUI solutions there are distances fixed in guidelines for the operating system - for mobiles we recommend to put the button in a second line (this is preferred against putting the button close to the input field).
- Background Colors (Table): The alternating rows should vary decently and be preferably in slightly different shades - the selected colors are “eye bending”.

As described above, we optimized the process so that now already the story itself was usability tested. This shortened the feedback loop and reduced the necessary rework. Figure 8.6 shows such a story card together with the actual implementation.

8.3 Usability Tests

Usability tests are carried out to evaluate the running prototype. One of the usability studies was executed in January 2008 with ten respondents using a mobile phone. The classical task-based usability test method was used [Rub94]. Each respondent was asked to execute five different tasks. Tasks were carried out on a Nokia N95 mobile phone. To gather general feedback and general opinions, two interviews were carried out: One before and one after the task session (pre- and post-interview). Each task was accompanied by task specific post-questionnaires. An interview session lasted about one hour. For these tests the device’s standard browsers as well as opera-mini 3.0 were used as they provide different navigation options. The former is incorporating a PC-like mouse pointer while the latter uses a link marker to navigate through the interface.



(a) Story card

(b) Application screen.

Figure 8.6: Optimized version with Usability tested stories.

Additionally, respondents had to judge three different visual design paper-prototypes. We used the AttrakDiff questionnaire [HBK] to capture the attitudes of the users towards the application in terms of graphical design, enjoyment, and aesthetics.

8.3.1 A Task Example

Task: Find the detailed description of a given movie, write a comment and rate it.

The task was completed without any greater difficulties by all respondents. On the "Home" page and on the "Channel" page respondents used the heading to find the detailed description and the video's thumbnail to watch the video. The respondents also did not encounter many problems on the "Clip Detail" page. The prominent position of the links "Comments", "Rate" and "Tell a friend" on top of the description page (see Figure 8.7) helped the respondents to understand which possibilities were offered.

On the "Clip Detail" page there were two interaction paradigms: Clicking on the link "Tell a fried" opened a new page, while the functions "Rate" and "Comment" were placed at the bottom of the "Clip Detail" page and users had to scroll down or use a link to jump down. The "Tell a friend functionality" caused no problems, but the task uncovered that on the mobile interface respondents did not recognize that they were scrolling down the page when using the anchor-links "Comment" and "Rate". To get back to the top of the site they pushed the "back" button.

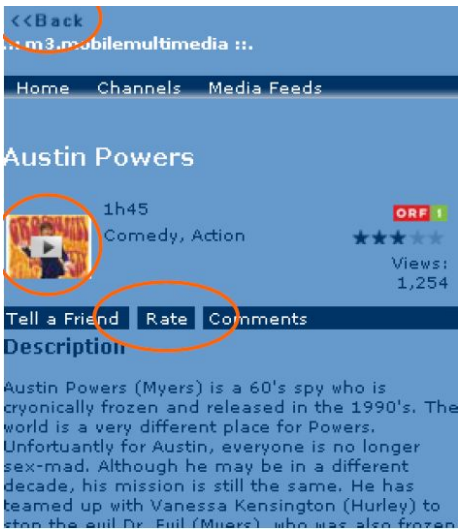
This did confuse some of the respondents as they jumped back to "Home" although their intention was to get to the top of the "Clip Detail" page. User comments also indicated that the longer the list of comments is, the more uncomfortable the site is to browse.

Respondents' Feedback/Comments:

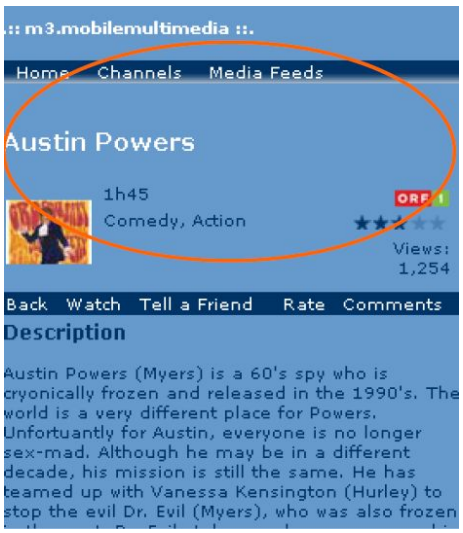
Through this task we got the following feedback and comments form the respondents:

- All respondents indicated that in their opinion the "Clip Detail" page provides a good overview.
- The design of the "Comments" and "Rating" section is good and intuitive. Too many comments on one page should be avoided as the page would get too long (1 respondent).
- Comments should be ordered in chronological sequence, beginning with the most recent entry (1 respondent).
- The space on top of the "Clip Detail" page (heading) should be used in a better way. This would provide more space for description texts (1 respondent).
- It should be possible to select which information is sent to another person via the "Tell a friend" function (the video's description, the video itself, etc.). Radio buttons should be used to specify one out of different possibilities (1 respondent).

Suggested Improvements



(a) Group and align interactive functions to the left.



(b) Use the space on top of the "Clip Detail" more efficiently.

Figure 8.7: Suggested improvements of menu and layout arrangement.

As a result, the following suggestions were made:

Links to "Tell a friend", "Rate" and "Comments": These elements describe "interactive functions" on the site and therefore should be kept together and aligned to the left side of the page.

Interactive functions ("Rate", "Comments", "Tell a Friend") : Incorporate them on a separate pages to avoid the confusion caused by in-page anchors.

Back Button: A dedicated back button should be integrated on top of the page. This is where basic navigation elements are expected.

Watch Button: A "watch" button should be designed and integrated consistently. An additional watch button – if necessary – should be placed on a particular spot on the site and not be integrated in the navigation menu. The "Watch" button should be visually highlighted.

Figure 8.7a shows the recommended menu layout and arrangement. Figure 8.7b shows the space on top of the "Clip Detail" page which should be used more efficiently.

8.3.2 AttrakDiff Questionaire

After the tasks, three designs were shown for ten seconds to the respondents and an AttrakDiff questionnaire was filled out for each one. This questionnaire allows to decompose the perception of a product and measure attractiveness, hedonic quality (stimulation and identity), and pragmatic quality (manipulation). These qualities are subjective and independent and their combination produces different product characteristics [HBK].

The following four types of quality are measured and described on a scale from -3 (poor quality) to +3 (good quality) [ML10]:

Pragmatic Quality: Describes the usability of a product and indicates how good a product is usable (in the user's sensation).

Hedonic Identity: Indicates the level at which users identify themselves with a particular interactive product.

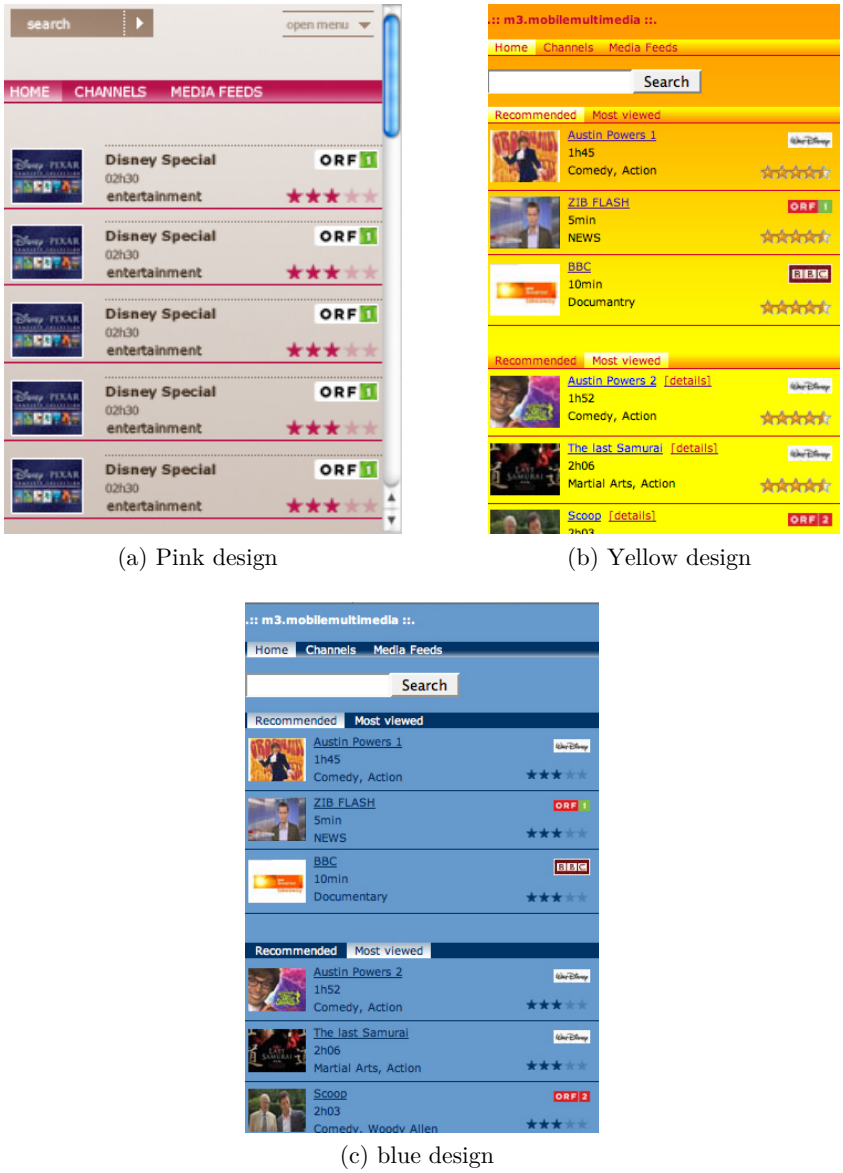
Hedonic Stimulation: Gives people the sensation to support them in gathering knowledge and in executing tasks.

Attractiveness: Is an overall category that shows how motivating, stimulating and likable a product appears to a user.

	Pink design	Yellow design	Blue design
Pragmatic quality	1	0,91	1,03
Hedonistic quality – Identity	-0,23	0,57	0,5
Hedonistic quality – Stimulation	-0,73	0,93	0,01
Attractiveness	0,99	2,29	1,87
Sum	1,03	4,7	3,41

Table 8.1: AttrakDiff results for the three designs.

From the mock-ups of the three different designs shown in Figure 8.8, the AttrakDiff



(a) Pink design

(b) Yellow design

(c) blue design

Figure 8.8: The three designs used for evaluation.

results presented in Table 8.1 shows that the yellow design was the most liked by the respondents. However, it was suggested that we could use the blue design as intended but the following improvements should be made:

- Accentuate contrast on whole site.
- Avoid light blue text on darker blue backgrounds.
- Introduce visually attractive design elements that increase the attractiveness of the site.
- Eliminate monotony by introducing more colors.

8.3.3 Test Results - Improvements of Layout and Design

The results of the tests suggest two main areas of improvements: the visual appearance and the usability of the prototype.

Concerning the visual appearance of the site we found that the menu, and navigation layout of the site was not optimal. Through the use of the color blue as text color and background color at the same time, equal text sizes throughout the interface and different alignments, the site's hierarchy was not visible for users (see Figure 8.8c). Additionally, the actual layout did not incorporate visually attractive design elements and was rated as pragmatic but monotone with a lack of stimulating elements (see Section 8.3.2).

Concerning the usability, it was suggested that on the "Channel" web page, a web-like calendar function to select dates should be integrated as the present function would not be usable for greater amounts of data. Additionally, all navigation menu elements should be separated from content menu elements ("Home" vs. "Watch"). Furthermore, interactive elements ("Rate", "Comment" etc.) should be placed on separate pages instead of on the bottom of a description page.

For the further development of the prototype the sub-site "Media Feeds" should be separated into two categories introducing the sites "create Media Feed" and "watch Media Feed". Special attention should be given to feedback mechanisms which at the moment do not support the user (feedback of search queries, display of media feed search results).

During this study, two of the developers were participating as observers. This was a great opportunity for the developers to see how users actually reacted to simple items and controls used in the interface and imagined their usage. It gave them a chance to realize how the end-users perceive the application, their feelings, and what they want from the application. At the end of the study, many new stories were generated from the observations and interviews with the end-users.

8.4 Review of the Used Instruments

Within the duration of this project we have extended traditional XP methods with knowledge derived from different HCI instruments. The novelty of our approach lies in the fact that we do not only use one or two usability methods to integrate them into the XP process, in fact we selected five instruments to enhance the existing XP process. This multi-instrument approach was developed to solve all the problems as introduced in Section 3.3. The five instruments we relied on are Extreme Personas, Usability Tests, Usability Expert Evaluations, Extended Unit Tests, and User Studies.

In 2009 a one-day retrospective workshop was conducted which was attended by all the team members and the usability engineer to reflect on the integration process as well as on the HCI instruments after introducing them at the start of our project. In the following sections, we provide a review of some the methods employed together with an explanation of our adaptations and suggestions for further improvements derived from this workshop.

8.4.1 Extreme Personas in Our Project

The Personas method should enable an end-user focused mindset to be established very quickly and hence should solve the problem of the development focus on the technical part. Additionally, the Personas should help to orient the project towards on-site customer *AND* end-user [HMS+09]. During our project we concentrated very much on the customer centered design process as well as the design process itself and nearly neglected the Personas.

There have been several issues and discussions on Extreme Personas during our project. First, the initially developed Personas were not satisfactorily distributed to either the development team or the customer-on-site by the usability engineers. Second, the development team did not give much credit to the two Personas which were provided [HMS+09]. From the point-of-view of the development team and the customer the main cause for this issues was that especially one Persona was so funny that they did not take it serious

It was concluded that Personas should be properly introduced to the team so that they are present consciously or unconsciously in the minds of the team members during planning, developing or undertaking any decision process. To achieve this it is important that the whole team and especially the customer learn to know the Personas and actively use them. Personas should also be kept alive during the development. This can be achieved by having them consciously in mind during story creation as well as using their names in the User Stories and during discussions.

Suggestions and Improvements

First, Personas should be introduced like new team members: You wouldn't ignore them. The introduction of Personas that will accompany the whole team during the development cycle and beyond is as important as the introduction of team members. Therefore, they need "room" and "positive energy". They should give the developers and customers a feeling of producing something valuable for someone who they like. Of course, fun can (and should) play a major role within teams - but be careful not to mob a persona!

Second, sometimes technicians think that it does not matter for whom they develop. The design is supposed to be the design-departments decision, the scope of implementing which feature is part of the customer's work, and over-all everything is pre- and post-tested by the Usability Engineer anyhow - so why worry? If this would be true, why did all technicians in our development team take part in a months-lasting discussion about the user-group we were developing and producing for? So even if these opinions are stated, it is our experience that developers, despite of their statements, do care about these issues and therefore the introduction of Personas will be beneficial.

Especially from the part of the customers, the opinion is strong that the technical or business-related decision processes are colored by conscious and unconscious inputs. That is why Personas should be present in an appropriate form and should have their own stable place around the developers. Stories and features should be developed for the Personas and their names should be used on the story cards and

during discussions. This will help them stay alive and influence the team.

Therefore, the advice from the customer-side is to make Personas available and visible, take part in the process of developing them, introduce them to the team, and have them (consciously!) in mind when planning or undertaking any decision process.

For the Introduction of the Personas, role playing games are a simple and effective method. It allows the team members to get familiar with the characteristics of the Personas and to fill in gaps in their "personality" during the play. This, in conjunction with a story out of the Persona's "life" which illustrates some of her features, makes the Persona more "alive", "real" and easy to remember. These games are fun which makes them rememberable and adds a positive feeling, requires the team members to think as the Personas, and allow them to experience the Personas in reality. When a Persona undergoes a greater refactoring or another Persona is introduced, it is recommended to "explain" this with a new role playing game.

8.4.2 Usability Tests in Our Project

In our project, usability tests in the laboratory have included end-users as demanded by the UCD process but not explicitly demanded by XP.

We conducted a formal usability test with 10 end-users in January 2008 in the usability laboratory of CURE. This test was also attended by two developers as observers [HLM+08c]. We noticed that the usability tests had impact beyond the expected (which is giving input for the design).

We saw that the mindset of developers changed dramatically when seeing real users handling the application during these tests [HLM+08c][HMS+09]. The developers who attended the usability tests got more biased towards user-centered thinking than the others [HLM+08c].

When it comes to the results of the tests there was an agreement that the tests were too early in the project to tell us a lot about the usability problems of the application. The system was very fast moving at that time, changes in features were high because of new demands from the stakeholders. Furthermore, the reporting period was too long. When the report arrived there already had been so many changes in the application that many recommendations were already obsolete [HMS+09].

Unfortunately, we could not conduct usability tests frequently with end-users due to time and budget constraints but mitigated it with usability expert evaluations.

Suggestions and Improvements

Usability tests are expensive and time consuming because of the recruiting and inclusion of real test-users. But according to Nielson and Landauer [NL93], five participants are sufficient to find most of the usability issues. They developed a formula, $Nf = N(1 - (1 - L)^n)$, which allows to calculate the sample size of participants for usability tests, where N is the total number of usability problems, Nf is the number which can be found, L is the percentage discovered by a single

Users	% Found	% Found	% Found	% Found
1	10	20	31	50
2	19	36	52,39	75
3	27,1	48,8	67,15	87,5
4	34,39	59,04	77,33	93,75
5	40,95	67,23	84,36	96,88
10	65,13	89,26	97,55	99,90
15	79,41	96,48	99,62	100,00
20	87,84	98,85	99,94	100,00
30	95,76	99,88	100,00	100,00
40	98,52	99,99	100,00	100,00
50	99,48	100,00	100,00	100

(a) Likelyhood of usability problems detection of different severity calculated according to Nielson [NL93].

Users	Minimum %	Mean %	SD	SE
5	55	85,55	9,30	0,93
10	82	94,69	3,22	0,32
15	90	97,05	2,12	0,21
20	95	98,4	1,61	0,16
30	97	99	1,13	0,15
40	98	99,6	0,81	0,11
50	98	100	0	0

(b) Test results with real, randomly selected groups of different sizes [Fau03].

Table 8.2: Relationship between the number of test-users and found errors.

user, and n is the number of users. The formula states that for severe usability problems a small number of users is sufficient (see Table 8.2a). Based on studied projects, Nielson assumes L to be 31% [Nie00]. The assumption of the number five for the size of the test-user group is origins from the fact that Nielson recommends iterative UI testing and assumes three iterations for each user, which is thought to be equivalent to 15 users which would find over 99% of all problems [Nie00].

This statement was challenged in other experiments where different results were reported. Falkner [Fau03] conducted usability tests with a group of 60 people were randomly groups of different sizes were selected. The average result for a group of five people as shown in Table 8.2b supports Nielsons claim. However, the minimal number found is far below. Although Nielsons statement has not proven to be true in all cases, it is still a valid assumption for the general case. Especially if we consider an iterative approach and the accumulation of the results, as the numbers show that for 15 users also in the minimum case the detection rate is 90%. Furthermore, Falkner gives no indication about the severity of found Problems, but Nielsons proposed detection rate increases with the severity.

We recommend smaller tests after every few iterations of the application (better for every iteration) or at least after every release. To keep the costs feasible and

since the system in an agile project can be a very fast moving target, not always the entire system should be tested and the number of test-users per test can also be limited to one to two persons. The low number of participants can be compensated by the increased frequency of tests resulting in a similar coverage than a big test (see Table 8.2). Important here is to switch users and experts frequently and to test already finished functionality repeatedly to achieve a higher coverage and similar results. This implies that it has to be tracked which functionality was already tested and how often to set the focus of the tests and to accumulate the results. Bigger tests should only be made when no major changes are expected and the system is quite stable [HMS+09].

Furthermore we recommend that all team members watch such tests frequently to get a better insight into the users' problems. The user tests provide feedback about the UI and reveal the users' mental model of how they expect the system to work. This is especially important for the customer because many important stories can immediately be derived from these observations. It also creates a user centered thinking which helps when inventing new stories. As watching the whole tests can be time consuming, especially when frequent tests are made, highlight videos of the tests should be produced containing only the relevant scenes.

8.4.3 Expert Evaluations in Our Project

Usability input is needed at different times: when writing UI related stories, before or during implementation of the stories, and after implementation. In our project, usability expert evaluations are done by the off-site usability engineer, usually in the form of an ad-hoc input. The communication uses different channels, like instant messaging, email, and video-conferencing, depending on the urgency and form of input required. The results of this continuous communication are far less usability-fixes.

In our process the customer writes stories together with a developer. When a UI related story is written, it is sent along with the refined paper prototype to the usability engineer at least three days before the iteration planning. The advantage is that during the iteration planning we already have an usability tested story.

When technical questions arise during the implementation – for instance that a certain demand from the usability side would cost too much, it is advised to call the usability-engineer or have a short video-conference. Here it is important to have the feedback as quick as possible. For us two hours to one day delay is acceptable for this quick feedback from the usability engineer.

After implementation and when the system is deployed the usability input is not pressing. It can be delayed for days or even weeks to get usability feedback.

Especially for web applications, but also for other projects, it is advisable that the UE has always access to the latest version, which can be achieve easily with a continuous integration system.

During our project, it took some time until the customers embraced the possibility of asking the advice of the usability engineer in advance. This might have been caused by the increasing trust in the usability engineer over time. After a while the

customers have reportedly valued the usability engineer's input higher. Another reason was the improved planning method which allowed preparing stories early enough to have time for usability input.

The experience with expert evaluation was a very positive one if the results arrived on time. For an XP project the usual way expert evaluations are done is not ideal. Instead of big long lasting application wide evaluations, smaller and faster evaluations on story level are needed. If the evaluation result comes after the story completion or iteration, the likelihood of ignoring the input increases dramatically. The reason is that either the input is already outdated due to the quick changes in an agile project, or other stories are higher prioritized by the customer. Consequently, a stripped down version of the usual expert evaluation process is more practical.

Suggestions and Improvements

If possible have a usability engineer in your team. The best would be a co-located engineer, but if the engineer is remotely located be sure to establish a flexible enough schedule and proper communication means.

User Stories should be prepared well before the actual planning. During this preparation when the story-cards are written, all the mock-ups should be drawn as well (use drawings by hand or simple drawing tools). This gives time to send the stories and mock-ups to the usability engineer for feedback [HMS+09].

The usability engineer should be trained in writing stories with the help of one of the developers to give his feedback in the form of stories, along with wireframes (drawings of individual pages to show the basic components) when he thinks they are needed.

Feedback from the usability-side should be quick. For story evaluation prior to iteration planning we suggest a maximum of 3 days for one-week iterations. For technical questions during the implementation the response should be within hours.

Thus, it is our advice to involve available usability engineers into the planning process as early as possible. Use her time and input only when it is intended to implement the results or when it is critical for the development. Be careful: Do not shift the evaluation and fixes to "when you have time", because this will never occur and thus no serious usability input will be realized.

8.4.4 Lightweight Prototypes in Our Project

We make use of two different types of mock-ups; low fidelity paper mock-ups and high fidelity mock-ups when needed; and get them evaluated by the customer and afterwards, send it to the usability engineer for additional feedback [HMS+09].

In our context, low fidelity paper mock-ups means rough sketches on paper or whiteboard which displays the important features, while high fidelity mock-ups mean an actual implementation of the page layout without functionality. We found that, if high fidelity mock-ups are really needed, it is more convenient to implement them instead of just using a drawing or an HTML mock-up. This is due to the fact that the resulting mock-up shows the real look of the page and its final version can

already be used inside the system.

As both the developers and the customer have been increasingly gaining knowledge about usability engineering, creating and evaluating paper prototypes is mostly sufficient. However if more detailed feedback is required, which usually concerns mostly graphic design issues, dummy implementations are made.

Suggestions and Improvements

Don't waste time while creating mock-ups! First, most of the time low fidelity mock-ups are sufficient. They can be drawn and changed rapidly by hand and contain all the important information. If you have to put them in electronic form (e.g, to send them to the UE) just use a camera.

Second, if a high fidelity mock-up is really needed (ask yourself twice!) try to create an empty implementation instead of using a drawing. Most probably it will take the same amount of time, but the results are more accurate and reusable. During this implementation you already find out if the intended layout can be done and when the mock-up is finished you can reuse the code.

8.4.5 Extended Unit Tests in Our Project

Our approach extends the XP unit tests by adding usability-specific test cases. Code based tests are enhanced with semantics to achieve this goal. For example code based tests can check against guidelines like the usage of capital letters on buttons or the correct label of a button. When adding semantics (the correct label of a button), we can include the test into the set of unit tests already used in XP.

Test- driven development in XP means to write tests first. These written tests then define the behavior of the application. Adding usability related unit tests with semantics allows us to define the usability of the application.

Unit tests – by definition – test small definable units of the software. The problem of patchwork application suggests using a holistic approach to testing. Therefore, the unit tests are extended by tests, which go beyond single units, and test complete interaction flows [WTS+08][HMS+09].

Suggestions and Improvements

As stated above, there are problems with automated usability tests. The current frameworks don't support a generalized approach and the available tools are not intended for inclusion into a continuous integration system. Therefore, we were only able to use a very low level testing approach of limited scope.

Nevertheless, our suggestion is to use also usability related unit tests as far as possible. But be aware that because of the insufficient frameworks this might be cumbersome and these tests may be prone to layout changes.

8.4.6 User Studies in Our Project

User studies conducted by usability engineers give insight into the many issues. What type of content would users like to see? How much time are they willing to spend viewing clips on mobile phones? Is the small screen size sufficient? What are the target groups for this service? What are the perceived limitations of the system? What would be suggested improvements? To what extent Web 2.0 aspects (ratings, recommendations, forums, etc.) are demanded and used?

We used user studies in the form of laddering interviews and field studies. In autumn 2007 the laddering interviews were conducted; the results were published in [LWST08]. The results were merged into Personas [Coo04] to have a visible representation of the intended customer group during development and were also used for continuous system improvement.

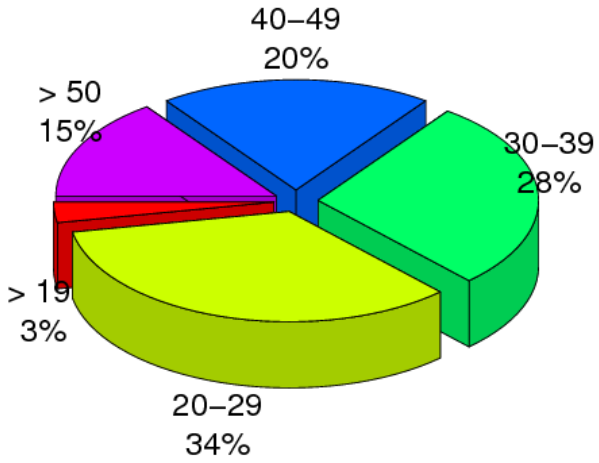


Figure 8.9: Trial Age group distribution.

From December 2008 till May 2009, a large field trial study was conducted with 185 real end-users spread throughout Austria. The participants used our application on mobile phones which were provided by our partner company and were pre-installed with all necessary software. The company announced the trial via email to its customers and the devices were distributed to volunteers on a first-come-first-serve basis. 80% of the volunteers were male, 20% female, and we had a broad age range as can be seen in Figure 8.9. 148 of the participants were equipped with a Nokia 6210, the remaining 37 people got a Nokia N95 [cur].

The users were able to use their devices freely to access multimedia content and did not have any restrictions. They were only asked to fill in questionnaires sent to their mobile phones and reply to certain SMS. The trial study also included diary studies, contextual interviews, laboratory usability tests, and focus groups [HMS+09]. Currently, the results are being analyzed.

We have implemented user-tracking and feedback mechanisms already in the basic architecture. The interviews, diary studies, usability tests, focus group, and log file analysis results will provide feedback on not only how the end-users perceive the product but it will also allow to collect statistical data from their actual usage behavior. In this way we will be able to provide more insights about the integration of HCI instruments into our adopted process. We will also gain insights into the context of mobile multimedia usage. We aim to continuously optimize our process till the project ends and will share the knowledge gained to the agile as well as the HCI communities.

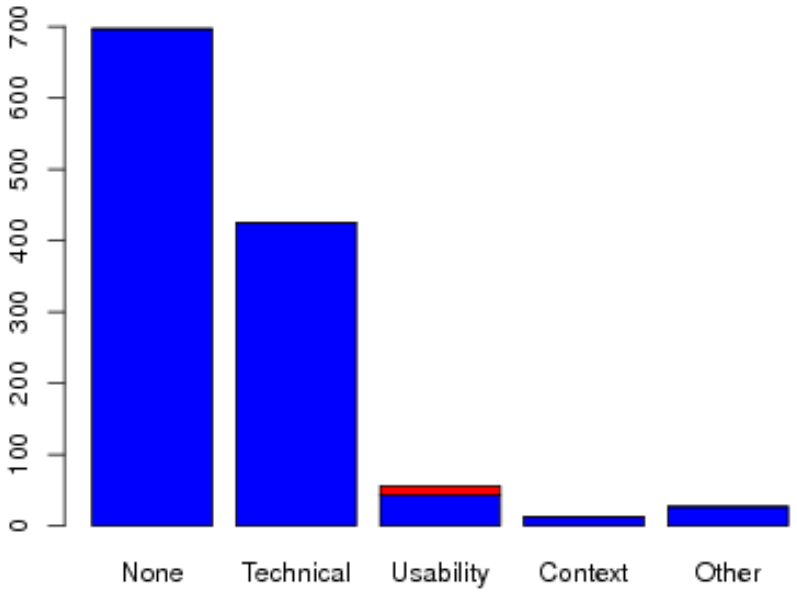


Figure 8.10: Problems during the trial. The application specific part of the usability problems is marked in red.

The already evaluated diary studies and SMS questionnaires which were conducted give us some clues about sources of problems for the users 8.10. These sources were categorized in four different categories: No problems at all, Technical problems, Usability problems, Context problems, and other problems.

All together there were 1219 responses evaluated. The majority of users (698) encountered no problems at all. This was followed by technical problems like no connection, insufficient bandwidth, problems when moving, etc. Context problems included lighting conditions, noisy environment etc.

Fortunately, there was a relatively low amount of usability problems reported (56), and most of these problems were not directly linked to our application but concerned device or browser specific features [Wol09]. Only 2.5% of the problems (13)

were actual usability problems of our application. Therefore, we conclude that our usability enhanced XP process worked well in practice.

This is supported by the findings of a focus group interview which was conducted by CURE after the trial [CK09]. The focus group was conducted to get a qualitative feedback about the trial and had eight participants, five male and 3 female. The age was between 20 and 51 years with an average of 35 years. All participants were technically interested and most of the participants work in a job with a technical or media background [CK09].

All in all, the good quality of the service surprised the users. The mentioned also other positive aspects like a well developed web site, the possibility to connect the system to a TV, different fields of application, a well designed user-interface, and quick respond and loading times. Statements concerning our user interface were [CK09]:

- "The user-interface was neat and well designed, the loading times were short."
- "I thought that the user interface would be much more spartan."
- "The optical impression is very well designed." (all users)
- "One search field was too less. A more fine-grained search or a better genre categorization would have been good."
- "Sometimes there were very much hits as result of a search query. "
- "There was no advanced search functionality."

Although the users had some suggestions for improvements, the overall consensus was that our system was well designed from the usability aspect.

8.5 Discussion and Conclusion

The integration of usability engineering methods works especially well because of the many overlapping principles (see Section 3.3). We use the previously described process since summer 2007 in our project which will end in 2010. So far, the responses from our users indicate that the process is working well and that it allows to deliver a working and usable system. The final usability tests will prove if the process is able to really enhance usability of XP style developed applications [WTS+08, HLM+08c, HMS+09].

Since summer 2007 the mentioned HCI instruments have been used in our project for enhancing the usability of the product. The usability engineer is well integrated into the development team. Until now we have learned a few lessons that are summarized here.

So far, the tight coupling of different expertise has led to high motivation among project members. Developers gain insight into the subtleties of UCD, HCI experts learn to understand the origin of usability problems. Especially the diverse technical testing frameworks demand technically aware HCI experts if they want to write their own usability tests. Depending on the used frameworks, the knowledge needed varies. In practice, this could become a problem when the chosen framework is complex and little time for learning is available [WTS+08, HLM+08c, HMS+09].

Continuous monitoring, evaluating and testing of the UI, and quick intervention can lower the danger of a patchwork-experience. Additionally, we could see that cultural problems between HCI and development seem to depend more on the involved persons than the methods used. Until now we did not experience these problems we assumed would come according to the literature.

Furthermore it was found out that especially ad-hoc input can be given sufficiently via mail. But especially for interface related stories it has turned out to be important to create and send mock-us, so that the usability engineer actually sees them rather than getting them described. However, we prefer to use synchronous communication between the project members if discussions arise.

Of course it would be the best case to have the UE co-located with the team or at least locally available, as face-to-face communication is more helpful in quickly resolving design issues. But the geographical distance between HCI practitioners and developers can partly be overcome by using e-mail, phone- or video-conferencing [HMS+09].

However, we find it important that also actual visits take place from time to time to improve the communication basis. This allows the usability engineer to see in which environment the application is developed and to get a better understanding of the developers and customers.

There are also various response times needed for usability input. During the story-writing process it is sufficient to get results within a few days. When quick fixes are needed or other input during an urgent re-planning, the usability engineer should be easy to contact for a quick advice via cell-phone or chat. The interaction with the UE early in the story creation-process results in saving time, increasing motivation, and gaining better realization of needed usability input early in the development.

When usability tests are conducted, instead of a big report of a formal usability test, the usability engineer should give report in the form of checkpoints which can be converted into stories quickly or even write usability stories himself. Therefore, the usability engineer should be trained in XP story writing to be able to deliver his findings in the form of User Stories.

One important point is the HCI awareness and training of the customer. As he is the responsible person for creating and prioritizing User Stories, he is the one to decide when usability input is needed and when usability related stories are scheduled. Therefore, proper customer and usability engineer coordination is necessary for enabling a good usability process in the development.

Chapter 9

XP Team Psychology - An Inside View

XP as a methodology for software development is now widely known. There are numerous case studies and reports of its successful application in real world projects as well as in the academic sector. But it is equally important to experience and report things that went wrong. This section is about problems and pitfalls that can occur when introducing XP. They are explained in detail, analyzed, and possible solutions are suggested.

9.1 Introduction

XP is a software (SW) development method which emphasizes pair and team work to a great extent. This means that the team should be more important to every developer than his own individual (ego) needs and habits. This can only be accomplished if all involved developers interact without problems and are focused in the same direction.

I am part of a team of PhD students which is working on a university project to research the XP methodology. We are developing a multimedia web application for the mobile phone market in a real business context and are investigating the XP methodology in this setting. As it can be seen there are two, possible contradicting, forces: business and science.

Our team consists of people with different cultural, educational, and technical background. They have different working habits, expectations and personalities. As each member is here for a PhD research, also the individual research goals are different. Introducing XP and working together as a team in this setting proved to be more difficult than expected.

In this section I will give an inside view into our XP team and describe and analyze the problems which we faced based on collected data as well as subjective observations and notes from reflection meetings (see Section 9.2.6). Possible solutions are suggested and a conclusion is given.

9.2 The Project Context

To understand the occurring issues it is necessary to see the context and background of the team and the project. In this section our scientific research questions are outlined as well as the team structure and our project goals for the actual

application.

9.2.1 Research Questions

Our project is about the research of the XP methodology. The project's scientific goal is to analyze agile SW development methods for small projects. Particularly, the project investigates to what extent very short feedback cycles with prioritized temporal scheduling, focusing on how simplicity, early customer involvement, automated tests, continuous code review, distributed code ownership, and continuous code integration with concurrent revisions and improvement of programs are influencing the quality and productivity of SW development. Further aims are the elaboration of a usability test procedure for mass market applications on mobile devices.

9.2.2 The Business Project

As an application and test object, a Multimedia Replayer for mobile devices is developed. Radio and TV programs (news, interviews, documentaries, talk shows, etc.) can be identified and retrieved by searching for words that have been spoken in the corresponding TV or radio programs or appear in a program's title: A kind of Google for TV and radio on mobile devices with speech to text capabilities. For this purpose we collaborate with various business partners and content providers[1].

9.2.3 Business Contradicting Science

Time pressure on the business side is likely to lead to a loss of at least some XP practices. Observed effects are among others that pair programming is reduced or abandoned, retrospectives are not held, pair switching occurs less frequently, unit tests are no longer written, and that stand-up meetings are skipped [WvD07]. So through the business pressure people could fall back into old habits which would heavily reduce the possibility to research the XP practices.

To overcome this possible problem from the beginning, it was agreed that the adherence to the XP practices has priority over business needs.

9.2.4 Team Structure

Everything really interesting that happens in software projects eventually comes down to people.

James Bach [Bac99]

This is especially true for a team-oriented method like XP. To get a better understanding of the occurring issues, it is vital to know about the team structure. Our team consists of six PhD students and a PhD supervisor. The students are five developers (from now on mentioned as D1, D2, D3, D4, D5), each holding a master degree in technical science, and one business person who also acts as our

[1]For more details see Section 4.

"On-site Customer" (see Section 9.4.7). Our supervisor is the project manager and sometimes also takes the role of the on-site customer.

D1, D2, and D3 are from the same country in Europe (Austria). Each one has already worked for some years as a professional full time programmer in a different company. The education of D1 and D3 is similar, as are their views on some issues but D1 often acts impulsive and sometimes more egocentric while D3 has more fixed views on certain issues but also seems to think more of the other people involved in the project. D2 is the oldest developer, has another educational and private background (interests and hobbies) and therefore different views (old school). This results sometimes in strong differences between D2 and the other two (D1 and D3). D4 and D5 are scholars from the same country in South Asia (Pakistan) with no prior experience in the software industry but with teaching experience. D4 is also the only female developer.

The cultural differences are very obvious in our team while the gender difference is not. D1, D2, and D3 are more active during discussions and during implementation while D4 and D5 are more passive and receptive. Especially during discussions it can be seen that the Asian culture is much more consensus oriented than the western culture and has a higher level of indirectness especially in work settings [SBLC+03]. This cultural fact is further enforced by the feeling of D4 and D5 that the others have more technical knowledge because of their working history.

The customer holds a master degree in business science and is mainly concerned with the business aspect of our project. He maintains the contact with our external business partners, is co-located with the team, and acts as on-site customer.

The project manager is a professor at our university who is equally interested in the science aspect as well as in the business aspect of our project. Mainly he only interacts with our on-site customer on business aspects, but also sometimes he plays this role himself. He rarely can be with the team because of his other duties.

9.2.5 The Project Cycles

The project started with an investigation phase in which the XP method was studied and the process was defined. The first test release took one month, the second two months and from the third release on we had a three month (quarterly) cycle for releases. Iterations were of one week duration for the first releases, but were changed later to two weeks in order to reduce the planning overhead.

We do release planning, iteration planning, as well as release and iteration reflections. These activities were planned to occur at the beginning and at the end of the respective cycle. However our reflections mostly took place at the beginning of the next cycle.

9.2.6 Data Collection

As we want to analyze the XP process, we tried out various ways of data collection. However, none lasted very long. The method of data collection as well as the used tools changed very frequently and it is not possible to get the whole picture of the history of our project from the available sources.

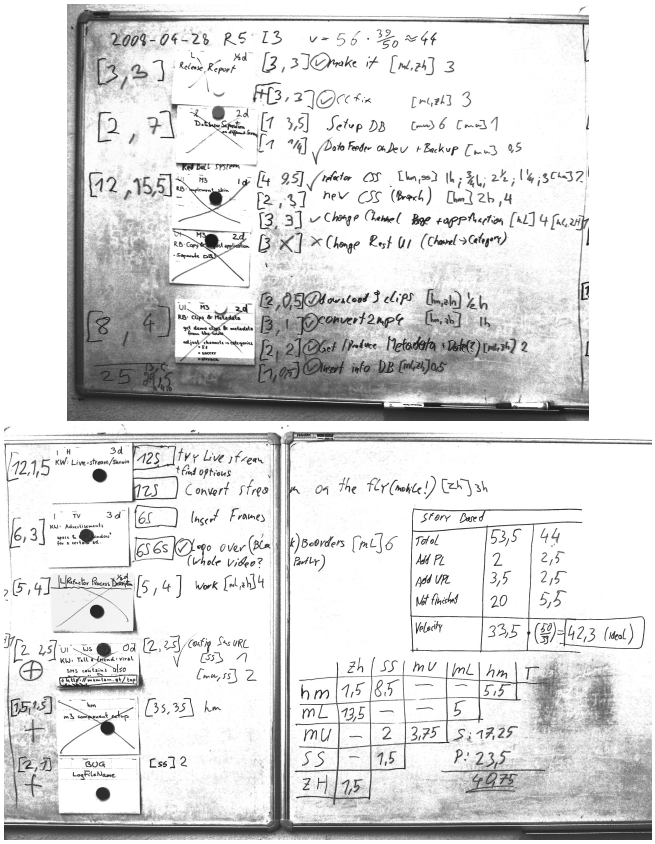


Figure 9.1: Data on whiteboards at the end of an iteration.

We started too “high tech” with different web based tracking tools, excel sheets, etc. But, either things were not fitting our needs, or it was too cumbersome to collect the data. Finally, we started to use a “low tech” method by tracking progress on the whiteboard and making pictures of it at the start and end of each iteration (see Figure 9.1). So we now finally follow the XP advice which states: “Don’t put too many things into the computer.”[BA04].

The only thing left to reflect the whole past of our project is our code base. Fortunately, since the very beginning of our coding activities we defined a specific format for log messages which allows checking which pair or solo developer made which commit to the repository. The data used here for analysis represents only the coding part of our repository. Commits in the other parts (science, business, etc.) where work was also done in pairs, are omitted. Again we (accidentally) follow here an XP rule: The code communicates [BA04].

Other sources of data are the iteration and release reflections. Here everyone talks about what went well or wrong in the last cycle. Also decisions to change some rules or practices are made here. Since three releases (9 months) we also use the XP evaluation framework (XP-EF) [LW04] at the end of each release. This allows us to compare our releases. For qualitative analysis, we use some data derived from the Shodan survey of the XP-EF which should capture subjective feelings about certain

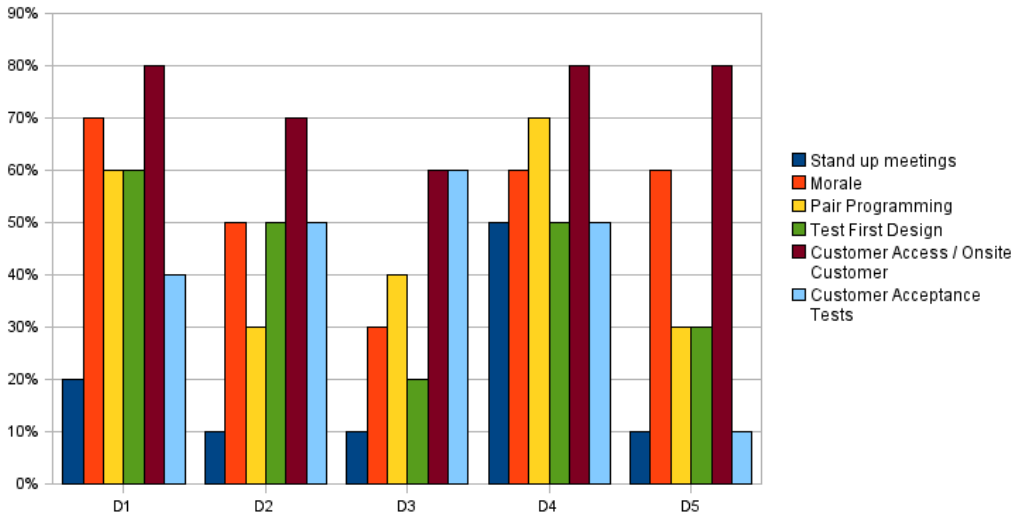


Figure 9.2: Shodan metrics of the last release per developer.

issues and practices (see Figures 9.2 and 9.3) [2] Figure 9.2 shows the different views of the individual developers (measured after Release 4) while Figure 9.3 reflects the overall trend of the opinions on some practices during the last three releases.

9.3 Process Issues

While trying to implement the XP methodology, we faced different problems. The general issues are described here in detail. It is also tried to analyze them and suggest possible improvements or solutions.

9.3.1 General

All team members have joined the project to do research for their PhD. They all want to bring something into the project, but all have slightly different ideas and goals.

Initially, there was the idea that XP should be used for everything including science tasks like paper writing, etc. It should be tried to use it as "pure" as possible in the beginning to experience its strengths and weaknesses. Modifications should be made only if, after some learning time, the process became familiar, clear, and was running smoothly. This was strongly opposed by some team members which were convinced that our setting is special, that the XP methodology is only suited for programming and that it has to be tailored immediately to the special needs of our

[2] The Shodan Adherence Survey is part of the XP-EF and is based on the XP practices dependency graph drawn by Kent Beck [Bec99b]. It consists of 15 questions which are answered by the team members individually and measures to what extend the XP practices are followed. These questions measure to what extend (0% = never 100%=always in 10% steps) the individual team member used a certain practice.

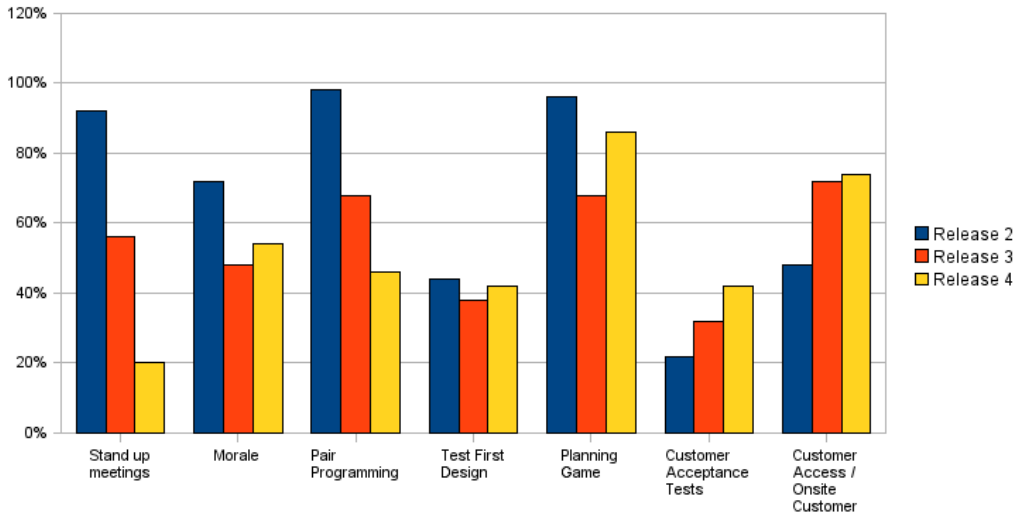


Figure 9.3: Shodan metrics of the last three releases.

special situation. And it is clear that with the strong conviction that "it will not work for us the way it is" it actually never did. These two forces were strongly opposing each other and led to long discussions with almost no results.

Also the general approach to the project was different. Some were eager to research XP and saw the application development as a vehicle for this, while others wanted to use XP and the development as a vehicle to learn and research other areas like Java, Patterns, etc. There was also the opinion that XP is so easy that there is actually nothing to research and discuss. Therefore, the different backgrounds and attitudes of the people caused problems.

Altogether this resulted in a general reduction of motivation. It can be seen from Figure 9.3 that in the course of the last three releases the morale – measured by the question: "how often can you say you enjoy your work?" – is not too high.

9.3.2 Leadership

The leadership style in our project was a very democratic and delegating one from the beginning. As XP is an agile method which focuses on "Individuals and interactions over processes and tools" [Man01] and the participants were PhD students which had to bring in their own ideas, this seemed to be a reasonable choice. Therefore, the team had no real leader but many different ideas and opinions which caused the democratic decision process to become long and tiring.

Here it would have been better to have a strict framework and a directing leader at the beginning. Especially during the time it takes to get accustomed to the new methodology. This would have provided a clearly structured setting of what to do and how to work, allowing the team members to get accustomed to this structure and each other. After the process would have been established, this strict form of leadership should have been changed allowing the team to perform on its own.

Although this seems to go against the agile principle of change and people centric approaches, I recommend using it for forming new teams. Feistinger has shown that "If a person is induced to do or say something which is contrary to his private opinion, there will be a tendency for him to change his opinion so as to bring it into correspondence with what he has done or said" [FC59]. This means that using a highly directive leadership style in the beginning the teams opinions and goals could have been aligned more easily.

However as the theory of loss aversion tells us that losses are perceived more powerful than non-gains [KT79], introducing this leadership style now would probably result in much heavier resistance. So in the case of an already established team other measures have to be taken. Overall a situational leadership approach as suggested in [HBJ07] is recommended.

9.3.3 Team Discussions

At first we tried to make almost every decision a team decision. But the different working background of the people led to long lasting discussions about how things should be done, mostly with no satisfying outcome.

This again was against XP principles. XP tries to avoid long team discussions because they decrease the team's energy [BA04]. But it was seen as necessary because we first had to define our own XP process according to the literature.

Another reason for the discussion culture was our scientific goal and background. As scientists we are trained to examine things in detail and to take into account all theoretically possible situations. Especially if we want to research something, we want to be sure that the method used is perfectly suited for all circumstances.

This level of detail is usually not necessary for practical application. Even on the contrary – XP tells us: "You ain't gonna need it" (YAGNI). Most exceptions we thought of – and discussed in great detail – never occurred during development. Also the discussion culture was not well established. Discussions had a tendency to drift to other points or became too detailed, needing additional time and energy.

Finally, we shifted from long discussion –which also can be seen as big design up front – to tryouts and small on demand decisions.

9.3.4 Agility

Agile methods invite and sometimes even force their users to modify them according to their needs. For example the available sources give no concrete XP implementation. Instead they list practices and values which also vary greatly between [Bec99b] and [BA04]. It is also stated explicitly in the literature that "There is no pure XP" [BA04], and that it has to be tailored according to the specific needs [Bro05]. However it is very important to be aware of what and why you change, because major sources for change are old habits. Changing a method to fit the old habits reduces resistance and makes it easier to follow this method. But most of the benefits that this method could provide are then lost as well as the chance of learning something new and valuable.

It is important that before you break the rules, you first learn and understand them completely. Changing a new method before it is familiar is a common but foolish behavior. Because: "Obviously any new use must feel different from the old, and if the old use felt right, the new use was bound to feel wrong" [Ale01]. Many "deficits" of the new method are just perceived as such because the method feels not familiar, or is not applied correctly – which needs time [Bro05]. Only after the new method is familiar and understood it should be altered. It is generally accepted that suppressing habitual responses is difficult and often not successful [AD00] but there is evidence that by forming implementation intentions old habits may be replaced by alternative behavior [HAL].

As Kent Beck states "Any one practice doesn't stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance."[Bec99b]. So we tried to implement the new method as a whole from the beginning, trying to force the developers to abandon all old habits at once. A better – and more agile approach – would have been to implement it in an iterative way, adding new practices only when the old ones became familiar and were working smoothly. It would have allowed us to focus on one practice at a time. Thus the amount of necessary change and the resulting resistance would have been minimized and the probability of success would have been increased.

9.3.5 Discipline

Simple, clear purpose and principles give rise to complex and intelligent behavior.
Complex rules and regulations give rise to simple and stupid behavior.

Hock, Founder and former CEO of VISA International

Agility allows changing the rules according to specific needs. But still it needs discipline to follow them. We did the changing of rules frequently as a result of our reflection meetings or discussions. However, the discipline required to follow them was missing.

The perception of the reasons differs from person to person: For some there were too many rules[3]. For some the rules were too strict. For others, the willingness and discipline to follow the rules was not there. Sometimes "agreements" were made just to end tiring discussions. Because of their informal nature, rules were forgotten quickly. The old habits were stronger than the new rule[4]. The rules seemed to be against common sense[5]. Some were content with an "almost" reached goal or just cared about setting rules and not about obeying them, while others wanted to reach the goal set by themselves.

This can be illustrated by the time schedule example: It was tried for almost three months to start at a specific time in the morning (8:30) with the stand-up meeting. The time was chosen by the team and everyone agreed. However, it was rarely the

[3]There were just a few rules, but it can be explained by the fast changing of rules - which made them appear more. Also the long discussions about how to handle specific border cases can contribute to this impression.
[4]Which prevents new experiences
[5]Which means contradicting old experiences

case that all people were present at that time. Delays of 5-10 minutes were the regular case. We even tried to enforce this time with team pressure[6], but with no effect. The perception of this behavior was different. For some it was unacceptable and was seen as a waste of team time, while others thought that the rule was working, but the interpretation – to be punctual on the minute – was way too strict.

Discipline is a personal issue, highly influenced by the individual goals. Different views on the project seem to be a major reason for the discipline problems. Not everyone sees the same aspects of the project as important and there is no consensus or greater common goal which could align the behavior of the team members.

9.4 XP Practices Issues

Apart from the general issues, there were also more specific ones concerning different practices. Although some of the root causes can be found in the general issues mentioned above, these issues are more specifically concerned with different practices.

9.4.1 Whole Team

XP states the "Whole team" as a key practice [Bec99b]. Therefore, the team has priority over the pair and the pair over the individual team member. It is contrary to most programmers' prior experiences in the field of SW development.

Although SW development is almost always done in teams, these "teams" mostly consist of individuals working at their own schedule on their own piece of code in their own style. In this definition of "team" the developer has many degrees of freedom.

XP with practices like pair programming, test-first design, and collaborative code ownership reduce some aspects of this freedom[7]. The own schedule has to be adapted to the pairing partner or to the whole team to be able to switch partners. The code is team property and the programming style is dictated by the method (e.g., test-first design) and influenced by the pairing partner.

These are major changes for a SW developer. And it seems that just thinking that people should be ready and aware of this when joining an XP team is not sufficient.

9.4.2 Co-location

We sit together in one room. According to XP it should help in knowing what other people are doing and in distributing information. This could be observed, however,

[6] We discussed this issue many times, made daily delay charts, talked about introducing small fines, and finally people brought cakes if they came late – which of course made coming late more accepted by the team ☺.

[7] On the other hand XP increases different aspects of freedom like schedule estimation by the developer, no overtime, etc.

it also turned out that this alone is not enough. While sitting in the same room, we often still do not know what the other developers are doing.

It can be partly explained by the increased solo time. During pairing the partners communicate. This gives information about the task they are working on, and especially about how they work, to the other team members. During solo work this flow of information is missing. To partly compensate the lack of information flow stand-up meetings are proposed by the XP methodology (9.4.3).

9.4.3 Stand-up Meeting

Stand-up meetings are amongst the most discussed practices in our team. As XP tries to avoid meetings, stand-ups are an exception. They should be there to augment the Informative Workspace and support the practice of co-location. Co-location provides information about "How" work is done while stand-ups tell "What" is done and what is planned to be done. So everybody knows all the time what is going on and is therefore able to help the others. The practice consists of a short meeting (5-15min) where every developer tells shortly what he has done since the last time, what he intends to do, and what problems he has encountered. It is hard to believe that something so simple can go wrong - but it does [Yip] !

There are various reasons why it is not working in our team. First the act of "standing up" (initially designed to keep the meetings short) was seen only as meaningless formality that was just followed because it was in the book. The second thing was that the meetings were used as start of the work. As not everyone was here at the same time, no work was done before the stand-up meeting, and the people who came earlier had to wait for the latecomers, which was a problem for some, but not all.

Another reason was that stand-up meetings tended to become mixed with problem solving, story telling, announcing unrelated stuff, planning, etc. So the purpose of the stand-up meeting – to be a short, general information point and to make and check commitments – was lost. It became boring and too long for the involved people. Because we all sit together it was the strong opinion of some, that actually the meetings are rarely necessary. They were perceived as repetitions of what was already known.

Although some of the proposed solutions from [Yip] were tried, it did not work out yet. So we decided to make stand-up meetings on demand. However, it is rarely done and is substituted by informal communication between team members. This seems to be sufficient but also seems to be less effective than a working stand-up.

9.4.4 Pair Programming

We have experienced most of the positive effects of Pair Programming (PP) stated in [CW01] and [WK02] in our project. We had better knowledge transfer, fewer bugs through continuous review, pair learning, etc. But from the beginning there were opinions that pairing is not useful for all tasks, and the general trend in our project is going from pair to solo work (see Figure 9.4). Also, if pairing occurs, it

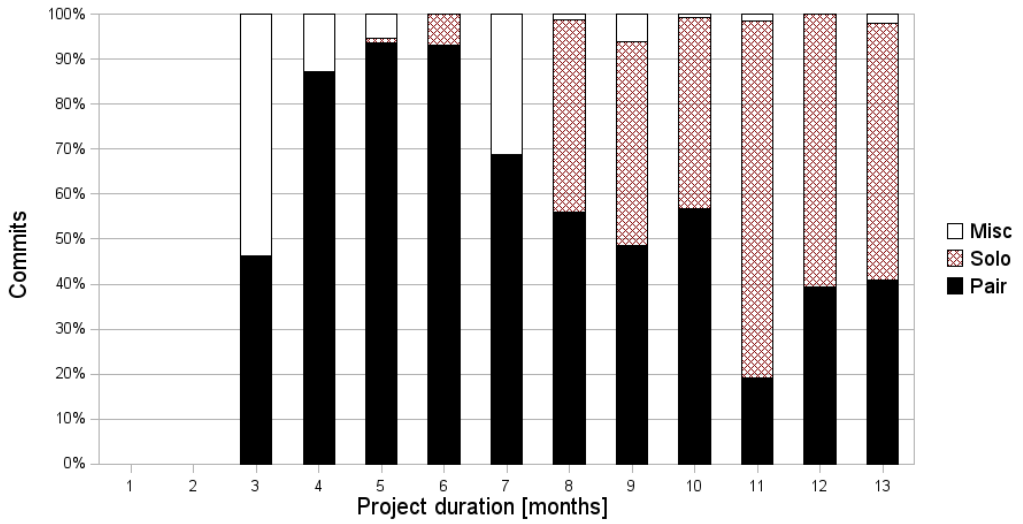


Figure 9.4: Distribution of commits.

is mostly the same people pairing, which can be seen in Figure 9.5 [8].

One reason for it is different points of view and previous long discussions. The knowledge about the different views of other people makes people reluctant to pair. To avoid more long discussions, pairing is avoided with certain people. This behavior corresponds to the theory of cognitive dissonance [Fes97], which states that dissonance is created by diverging opinions. One way of reducing this dissonance is to avoid certain situations or people – which is our case. Also the physical location is a factor as in our setting neighbors are more likely to pair. Our South Asian developers (D4, D5) sit in the middle and are more discreet. Therefore, they are the most likely pairing partners.

Another reason is the fact that it takes very long to start. Somehow it is hard to find a pair partner and really start to work. Mostly if someone wants to pair and announce it, nobody feels concerned. If a pairing partner is finally found, it takes additional time until the pair really starts to work. The same time delay occurs after breaks or interruptions.

Therefore, in our context pairing means waiting and a huge effort to start – which makes it infrequent. If someone wants to work, he just starts. This behavior is also strengthened by the previous work experiences. So the reasons for the increased solo work are old habits and firm convictions which are enforced by some of the new experiences.

It can be seen in Figure 9.5 that the pairs D1, D2 and D2, D3 are the most infrequent while the pairs D2, D5 and D3, D4 are the most frequent. There are several reasons for it. The strong differences between D2 and D1, D3 makes these

[8] At the beginning the log message format was still under development, so in Figure 9.4 there is a great part of "misc" commits in the first two months which can not be attributed to someone directly.

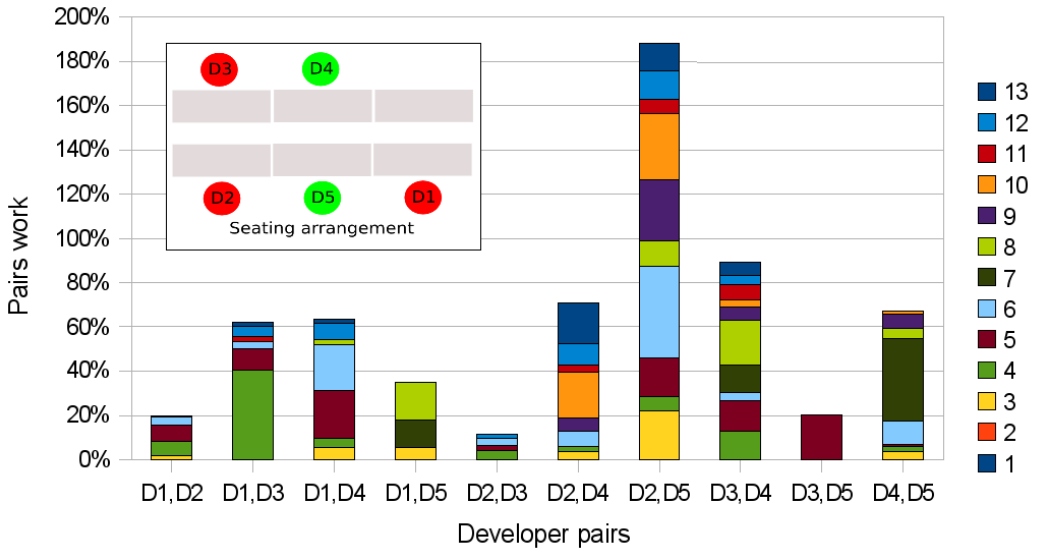


Figure 9.5: Total Commits per pair.

pairs unpopular. The neighboring effect: D2 and D5 are neighbors as well as D3 and D4. The cultural background of D4 and D5 makes them very adaptive and therefore they are often sought as pair partners. However, the active and dominant behavior of D1, D2, and D3 turns these pairing sessions mostly into solo performances with one partner only watching. This means that even if pairing occurs it is mostly not the pairing described in the XP literature. Here a combination of the "Professional Driver" (D1, D2, D3) and the "Too little ego" (D4, D5) problem, explained in [WK02], occurs. Both problems are enforced by the respective cultural background [SBLC+03]. Therefore, the knowledge transfer and learning factor are relatively low in these cases.

9.4.5 Simplicity and Yagni

These principles are in general among the most neglected in our project. Especially in the science part we always had long discussions about different exceptions which never occurred. But also on the programming side the perceptions of what is YET necessary to reach a certain goal differed strongly.

Simplicity is dependent on the point of view and experience of the individual. The different viewpoints and the different backgrounds are a huge factor for the strong disagreements of what is simple and what is needed.

It would be worthwhile to incorporate these principles because they reduce complexity and increase performance by maximizing the amount of work not done [Man01].

9.4.6 Planning Game

Especially in SW development, many projects are over time and budget or fail altogether [cha03a]. These facts are of course known by the customers. Therefore, it is

crucial to keep the commitments that you make, and that the process is transparent to the customer. XP tries to accomplish this by incorporating the customer into the development process.

The Planning Game is a very self similar aspect of XP which heavily includes the customer. The scope is refined at every step starting from release planning to iteration planning to stand-up meeting. You commit what you want to accomplish during a release, an iteration or a day.

While planning a day is left to the individual developer or pair and does not need additional backup data, planning for longer periods like weeks and months does. Tools are needed to be able to keep the commitments. These tools are the velocities which help to predict how much work can most likely be done. The velocity is based on "yesterdays weather" meaning you are likely to be able to accomplish the same amount of work in the next period as you were able to do in the last [BF00b]. To calculate the velocity the estimates of the accomplished User Stories are summed up, assuming the developers have a more or less constant estimation error.

By having different time scopes, release and iteration velocities have also different units of estimate. It means that one story has two estimates. A rough one for the release and a more detailed one for the iteration. We estimate in pair days for the release and in pair hours for the iteration.

It is crucial not to associate the velocity units with real time! To avoid this problem, [Bec99b] suggests estimating in arbitrary units like story points or gummi bears. We find that estimating in hours and days is more convenient, because we have a feeling for those units. However, still we sometimes experience troubles by confusing these times with the real times spent on the tasks. For example if we estimate five hours and work ten hours it looks as if our estimate is far off, if we would have estimated five gummi bears there would be no relation.

Planning at release scope is quite difficult. As a release lasts three months and we are just in the initial phase of interactions with different business partners, the customer often does not know what will happen and sometimes refuse to do release planning at all. Because of this fact the stories for the iterations rarely come from the bunch of stories selected during the release planning. This makes calculating the release velocity cumbersome and arbitrary. Also the result of the planning is mostly just valid for a short time because the stories and directions discussed are usually changed shortly after without re-planning. This means that most of the stories on the release board stay there and the main purpose of the release Planning Game – to give a rough direction for future work – is lost after some iterations. This leaves the developers without a vision of the future and results in decreasing morale.

Our planning practice for the iteration works quite well. We try to clarify the scope of the stories with the customer, write the acceptance criteria down and estimate the stories accordingly. But the actual implementation during the iteration mostly deviates heavily from planning. Sometimes the agreed acceptance criteria are changed, sometimes the YAGNI and simple design principles are violated resulting in extra work, and additionally there are often "small" or "urgent" customer demands to be done in between. All of this is mostly done without re-planning. The result is a low velocity and decreasing faith in planning.

This issue seems to be difficult to solve as it involves different aspects: First, the planning aspect which is seen as valuable by most of the developers. Second, the estimation and velocity aspect which is seen by some as overhead work because the values are not representing reality (because of inserted tasks, etc.) and because it seems to be mainly for the management which is perceived as using it the wrong way.

One possible solution would be to switch to gummi bears to avoid management confusions. It is also necessary to take planning more seriously and enforce more discipline than agility during iterations by trying to stick to plans or do explicit re-planning if changes occur. This would be easier if the release and iteration periods would be reduced to a better predictable time frame. A probably working solution would be to go back to one week iteration cycles and a one month release cycles. Therefore, we finally omitted release planning and went back to on week iterations with stories estimated in story points.

9.4.7 On-Site Customer

Our project is targeted at the mass market. Therefore, there is no real customer on site. To compensate for it, we have an external usability engineer who also performs end user tests, and who communicates with our "On-site Customer" which is our contact point. Our customer is in reality a customer proxy as he represents the end users as well as the business partners. However, he fulfills all the roles of a real customer: He is on-site, part of the team, writes User Stories, makes business prioritizations, and is available for clarifications during implementation [Bec99b].

At the beginning of the project the developers were also included and had a sort of customer role. This meant that decisions on what features were needed and what to implement were made in the team, leading to a long decision process.

XP states clearly: The customers specifies (What), the developer estimates (How and how long) [Bec99b]. This fundamental separation of concerns was broken at the beginning of our project. It happened that the developers in customer role needed a long time to find a common vision on the "What" – which most of the time also included already a big part of "Why". And then the implementing developers changed the already agreed solution according to their opinion, because they felt to have the right to do so, as they were also customers. This led to distrust and resignation.

To overcome this problem, we now have a dedicated customer (see Section 9.2.4) which has eased the situation but led to a different kind of problem. Developers take pride in the things they develop [DL99]. This increases the quality and keeps the developer motivated. Therefore, it is hard for them to implement solutions they are not convinced of, and they are tempted to alter them in their way, especially if they are also customers at the same time. To keep this attitude and still have the customer say "What" without destroying the developers' motivation if the opinions do not fit, the developer must focus on the "How". In the worst case it means being proud of the good implementation of a – from the developers viewpoint – totally stupid customer request. This is a necessary but hard to accomplish attitude change, especially if you care about the project and had the right to decide before

[KT79].

Another difficulty lies in the fact that the customer, being a business person, has a different point of view than the developers. Sometimes he just provides rough ideas where developers expect specifications, and on the other side the developers often do not understand the problems and issues the customer is worried about. The problem of the diverging viewpoints was solved by letting a developer and the customer write the stories together. This had a great impact on reducing the discussion time during planning, as the stories were mostly already small and concrete enough.

In general the idea of having a dedicated customer works pretty well and is also perceived positively by the developers (see Figure 9.3). Still sometimes problems in the decision making process occur. Our project manager is not always with the team but still sometimes act as customer. This can lead to the situation that the customers do not speak with one voice, and the developers get different information, as one customer may be overruled by the other.

9.4.8 Test-First Design

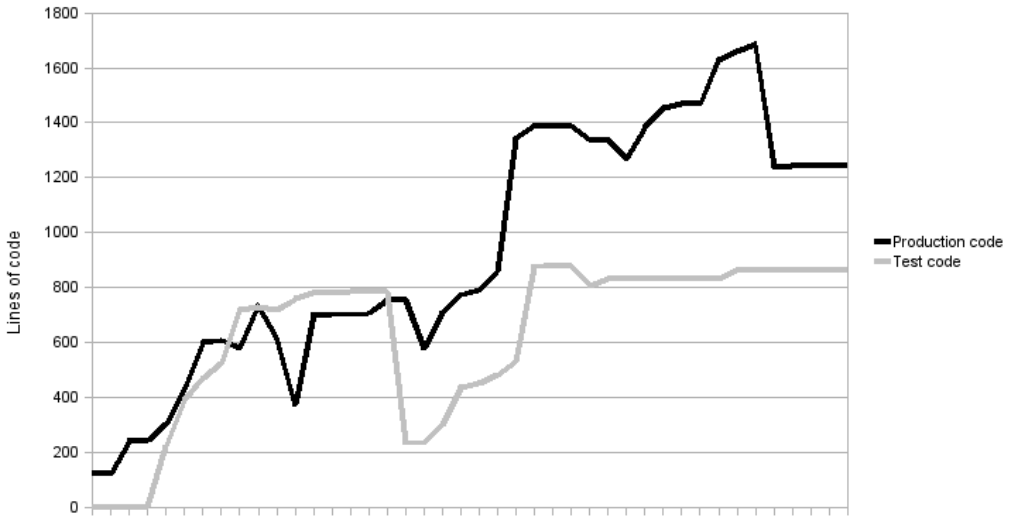


Figure 9.6: Test/Code development during the project.

Test-first design or test-driven development (TDD) is a very advantageous, but also very new way of developing for many programmers. Writing the tests before the code is the complete opposite of the traditional approach. While programmers are used to write an implementation, TDD forces them to first write an executable specification for the implementation. Therefore, starting programming in this way is very difficult for an experienced programmer because it means thinking in a completely different way than she is used to.

It can be seen in Figure 9.6 that especially at the start of the project, tests were usually neglected [9]. While it is better now, still many tests are written after the

[9]The huge drop in the lines especially in the test-code was due to a major refactoring where hard-

code. This practice still needs improvement, but is maybe amongst the hardest things to change as it goes against years of experience [AD00].

9.5 Conclusion

XP is a very promising methodology. However, its greatest strengths are also its biggest weaknesses in terms of implementation. Because of its flexibility it is easy and tempting to change it to the already familiar working behavior, and because of its people and team centralization it depends heavily on the smooth interaction of the people executing it.

XP consists of several practices. We tried to incorporate them all at once into our project. We overestimated ourselves and believed that because everyone involved in this project knew the goal (researching XP) and everyone works for a PhD, it would work. We thought that everyone had the same attitude towards XP and teamwork, and that studying the available XP literature would be sufficient to succeed in implementing this methodology.

However, we neglected the fact that teams have to grow and people need to get used to each other. Also that different PhD students have different views about the same projects and have to have different goals and topics. We also neglected the strengths of old habits and the necessity of an appropriate leadership style which could have avoided or reduced many problems.

The fact that XP itself is a highly self similar and iterative process allows us to think of a better way to introduce XP – in an XP way: Iteratively focusing on one practice and if it is familiar continuing to the next. The practice to choose should be the one with the highest "business value" which means the most beneficial and least controversial. This will demand only small changes in working habits at once and will allow the team to grow together. It will also reduce the amount of exhausting discussions which destroys the motivation. It will allow the individuals to gradually become a team and to work in the same direction.

Having experienced the slow decay of motivation and methodology over more than one year, we tried to make a new start by incorporating these lessons learned. We tried a step by step introduction which is recommended generally for the introduction of a new methodology. For this we attended an agile summer camp to get a common vision and to exchange thoughts with real-life XP practitioners. Afterwards we held a review meeting and selected the primary practices which needed improvement. We now do more pairing and TDD, as well as have changed the iteration planning process. Although the results are not perfect, our situation became better and is still improving.

coded test-data was reduced and extracted

Chapter 10

XP Context Factor Analysis by Mailing List Mining - a Preliminary Research

As stated above we observed different problems while introducing and applying XP. This raised the question if these problems are team specific or if they are commonly observed in the XP methodology. To answer this question, further investigations were made. The idea was to find and use data that are more general than the limited scope and relevance of individual case studies. For this reason the idea of data mining the XP mailing list was born.

Research on agile software development methods is becoming increasingly popular in industry and also gains much interest in research communities. But most studies are based on individual case studies, making them not generally applicable. To get the bigger picture an approach is investigated which uses the eXtreme Programming mailing list as basis data for analysis. The analysis identifies the important topics inside the community together with their weights, and their interconnection. For this investigation all messages from the year 2008 were evaluated. The results of this approach are presented and encountered problems as well as process improvements are suggested.

10.1 Introduction

XP is one of the best know agile methodologies [Sch08]. The methodology together with its set of practices proposed in [Bec99b] is therefore also in the main focus of researchers.

However, although many research projects have been carried out that investigated XP and its practices, the available data seems not to be representative. Most of the available data about XP is from anecdotal evidence, case studies, or experience reports. Many of these studies were done with students instead of professionals and often only one individual practice or a subset was applied. Also many studies focus only on one team of developers, often with their own customized XP variant, which makes the general applicability of the results questionably and a comparison not easy. Williams et al. have proposed an XP evaluation framework [LW04] to allow such a comparison, but it does not seem to be widely used. The problem of satisfactory data sources seems not to be easy to solve. On the one hand, the real-world application of the results with students is questionable. On the other hand, industrial teams are hard to acquire for participation and it is difficult to control the settings.

Therefore, another source of data is required to understand what is really going on in the XP community. For this purpose the XP mailing list [Yahb] is suggested. This list is heavily used by XP developers and is a readily available and huge repository which can give insights into the XP world. In this paper a preliminary analysis of the list is made using quantitative content analysis.

Online material like newsgroups, blogs, mailing list, etc. offer "a vast amount of new material for old questions" [Ire99]. Content analysis is one of the main techniques which are used to analyze this data. Social scientists use this method for the analysis and classification of social networks and the social behavior of the participants [FSW06]. It is also commonly used in formal educational settings [DWSVVK06], and has been an important topic of research for sentiment analysis and opinion mining [FRMF07][PL08]. Fortuna et al. used SVM classifiers to automatically detect message type and sentiment [FRMF07]. Bird et al. examined the mailing list of the Apache HTTP server project to determine the social structure of the developer network and to correlate list activity with changes in the code repository [BGD+06].

As data repository the eXtreme Programming mailing list "extremeprogramming@yahoogroups.com" is used. This group was founded on January 1, 2000 and has currently approx. 153,000 messages and over 9,000 members [Yahb]. The size of this community together with the huge amount of messages allows using the group as a basis for statistically founded conclusions.

Although the data is publicly available, there are possible privacy concerns [Ire99][PL08]. To avoid this issue, permission has been asked from the list owner as well as in the list itself (see message 147498 in [Yahb]).

In this preliminary study which considers only data from the year 2008, the feasibility of the data mining approach as well as the approach itself is evaluated. Furthermore, the experience from this evaluation is used for suggestions of how to improve the process.

In this paper first the methodology is explained, the individual steps together with the derived data are presented, results are shown, and finally a conclusion is made.

10.2 Methodology

Content analysis has been defined as "a systematic, replicable technique for compressing many words of text into fewer content categories based on explicit rules of coding" [Web90, Ste01]. It is a well know technique for offline material [LP02] as well as online content [Ire99]. There are qualitative and quantitative methods for analysis. For the XP group analysis a quantitative approach is used allowing a statistical evaluation. Quantitative methods range from simple word frequency counting over natural language processing and linguistics to time intensive manual message coding. This study uses the latter approach.

In his review about content analysis schemes for discussion groups, Wever points out that the choice of the unit of analysis is crucial because it will affect coding and comparability of the outcome [DWSVVK06]. For a preliminary analysis the complete message was chosen as unit.

Furthermore, it was decided that only the first message of a thread should be coded for content analysis in order to reduce the amount of messages which had to be manually processed. The resulting thread from this initial message was constructed and its properties were used for evaluation.

announcement article communication	question testing refactoring legacycode
announcement blog	question research
announcement event coding session	question resources design incremental
announcement meeting reminder	question specific design refactoring
announcement position application	question team motivation personality
announcement proposal certification	question testing acceptance howto specific
report pair	statement misc teaching
report planning team	statement apology doublemail
report team distributed scrum	statement listspecific satisfaction

Table 10.1: Example coding lines of messages.

In the "quantitative" approach the message is coded, summarized, and frequencies/percentages are calculated which are used for comparison and/or statistical testing [SMPJ06]. Messages were coded manually according their content. The first keyword (main category) describes the overall type of the message (see Table 10.3), the second keyword the overall topic, and subsequent keywords refine the topic (see Table 10.1).

This procedure resulted in a set of keywords representing category topics. This set was then consolidated in a following step. Overlapping terms were checked again and modified or unified if necessary.

Inter-coder reliability is a major concern in content analysis [DWSVVK06] as "Lack of reliability increases the probability of Type II errors (wrongly accepting the null-hypothesis)" [SMPJ06]. As only one coder was working, inter-coder reliability was not an issue.

For the statistical evaluation, some properties of the threads are used. The length of a thread is seen as an indicator for the importance of the topic. Based on the hypothesis that in online discourse a reply is usually associated with disagreement [FRMF07], the length is also used as a measurement for discussion [FSW06].

For assessing the degree of controversy of individual threads, the suggested discussion measure from [DLCA06] is used, where the number of authors divided by the number of messages gives an indication of the discussion activity. This measurement is between 1 and 0, with values close to one indicating no discussion and values close to zero a higher interaction level. For our analysis, this measure was changed to $1-\frac{authors}{messages}$. Now zero means no discussion, which seems more intuitive. We call this measurement "controversiality".

To account for the fact that topics which occur more often as starting topics of threads are obviously more popular, a weighted controversiality measurement was introduced. The measurement is called "w-controversiality" and is defined as $\frac{\sum threadcontroversiality}{\sqrt{numberofthreads}}$. W-controversiality indicates how controversial a topic is by taking into account how discussed it is and how often it appears.

This data was calculated for all coded threads and the keywords were sorted ac-

cording to the w-controversiality of the associated thread.

Furthermore we tried to get a context map of the interconnection of the topics, similar to the XP practice analysis done by Kobayashi et al. who analyzed the interconnection of XP practices during XP introduction [KKSP06]. While Kobayashi et al.'s data is based on developer interviews of two case studies and limited to the XP practices we have additional influential factors and a much larger amount of data. The context map can be derived from the coding, because the order of keywords reflects these connections (see Table 10.1) as it contains information about the *maincategory* → *topic* → *subtopic* relationship. This should allow determining environmental influences to the practices.

10.3 Analysis Method

The first step was to gather all messages from 2008 with the Yahoo Group Message Archiver (YGMA) [Yaha]. This resulted in 9,485 messages as plain text in MIME format. After the duplicate removal, 9,473 messages (about 6.2% of the total number of messages available) from 500 authors (5.3%) were left. As the manual evaluation was limited to initial messages of each thread (meaning no replies) this left 731 messages (7.7% of 2008 or 0.5% of all).

The initial messages were coded using the method described above. Then the codes were manually consolidated: different spellings were unified and overlapping categories were merged. After this procedure a graph of all messages was created. This was done using the "Message-ID:", "In-Reply-To:", and "References:" fields of the mail header. It has to be noted that depending on the mail client used, these ids are not always present. (E.g, 14 of the 9473 messages did not even have a message id and could therefore not be included into the graph.)

	threads	per author	per thread	thread/author	controversiality
initially	731	18.95	12.96	1.46	0.95
splitted	1423	18.95	6.64	2.85	0.95

Table 10.2: Overall properties of the 2008 data.

After this stage the graph was examined for topic changes and split accordingly (see Figure 10.1). This was done using a "cleaned" version of the subject line (artifacts like "Re:" were removed) which was checked against the parent. The splitting was done because a change in the subject would indicates a new discussion topic and thus the label of the root message no longer applies. This increased the number of threads from to 731 to 1423 (see Table 10.2).

These data was examined per main topic as can be seen in Table 10.3. For this examination the dataset was split per keyword, and for each keyword the sum of controversiality, messages, and threads as well as the w-controversiality was calculated. In the Table also the "was" and "answer" categories are included. The "answer" and "was" categories in the Table represent messages which were wrongly labeled as initial messages of a thread in the automated preprocessing step. The results were further split into threads with less than five and in threads with more than five messages, and as we are mainly interested in discussions, sorted by the

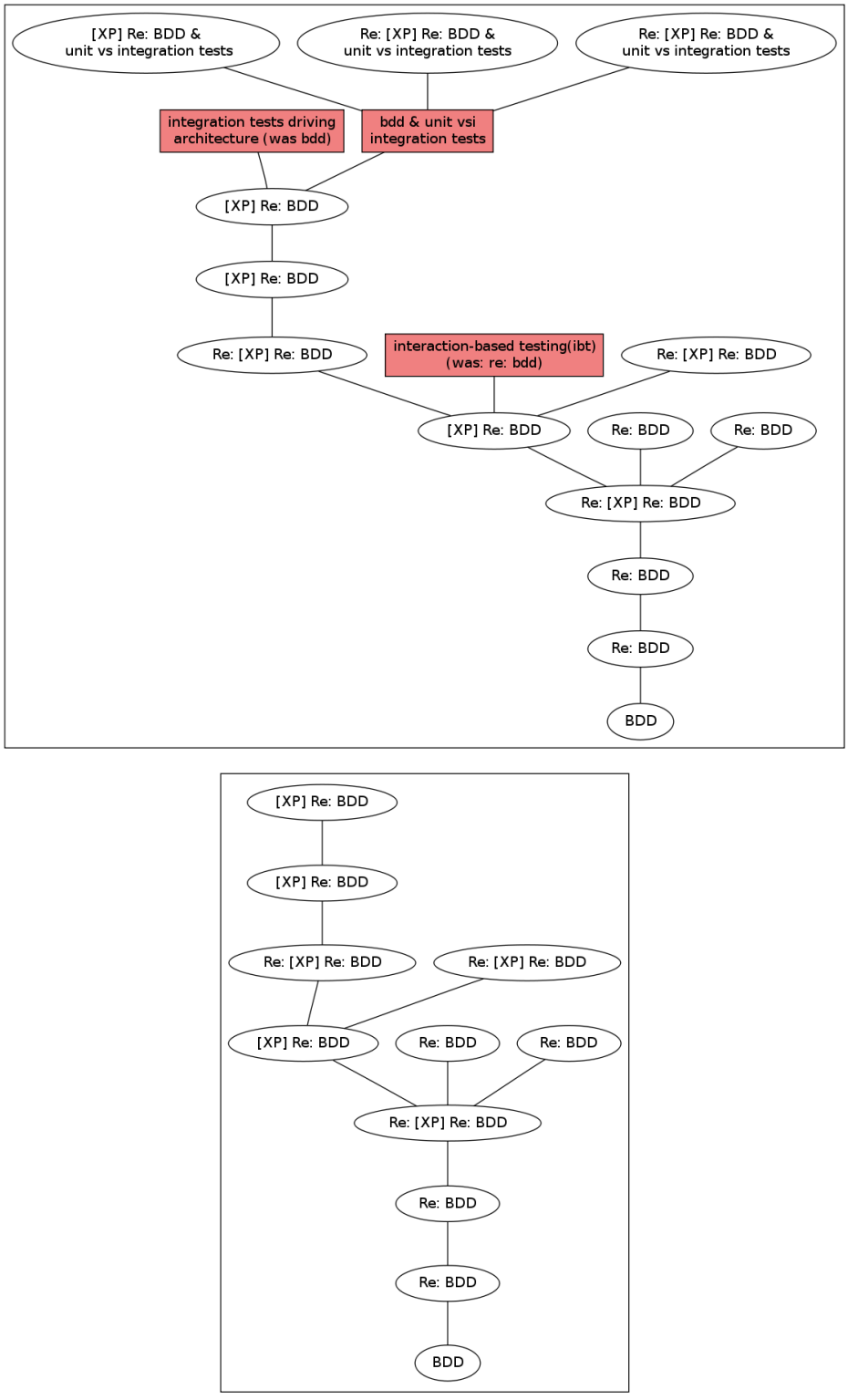


Figure 10.1: Example of thread splitting by subject (initial thread left and splitted right)

General Topic	Total messages				Threads with less than five messages				Threads with five or more messages			
	messages	**threads**	**Σ controversiality**	**w-controversiality**	**messages**	**threads**	**Σ controversiality**	**w-controversiality**	**messages**	**threads**	**Σ controversiality**	**w-controversiality**
not coded	3396	784	63.20	2.26	1082	649	7.92	0.311	2314	135	55.3	4.8
question	3122	211	60.27	4.15	179	75	5.50	0.635	2943	136	54.8	4.7
announcement	1458	241	21.52	1.39	278	201	4.00	0.282	1180	40	17.5	2.8
answer	662	100	14.96	1.50	121	72	2.50	0.295	541	28	12.5	2.4
statement	424	46	9.10	1.34	36	21	0.33	0.073	388	25	8.8	1.8
report	139	9	3.93	1.31	8	3	0.33	0.192	131	6	3.6	1.5
misc	150	11	3.16	0.95	15	6	0.75	0.306	135	5	2.4	1.1
was	95	6	1.56	0.64	5	3	0.00	0.000	90	3	1.6	0.9
spam	7	3	0.00	0.00	2	2	0.00	0.000	5	1	0.0	0.0
reminder	11	10	0.00	0.00	11	10	0.00	0.000	-	-	-	-
request	3	1	0.33	0.33	3	1	0.33	0.333	-	-	-	-
suggestion	3	1	0.00	0.00	3	1	0.00	0.000	-	-	-	-

Table 10.3: Comparison of messages, thread count and controversiality per topic for different datasets sorted by w-controversiality of threads with more than five messages.

latter.

It can be seen that the “question” category has not only the highest number of messages besides the ”not coded” threads, it has also the highest number of w-controversiality. The “announcement” category has the second highest ratings and even more threads than the “question” category. Therefore, we limit our further investigation to the “question” and “announcement” categories.

rank	keyword	messages	authors	threads	controversiality	w-controversiality
1	pair	181	72	9	4.121	1.374
2	testing	346	186	28	6.665	1.260
3	colocation	168	30	2	1.481	1.047
4	howto	130	56	7	2.714	1.026
5	team	269	108	10	3.004	0.950
6	resources	338	108	8	2.592	0.916
7	planning	99	54	11	2.900	0.874
8	specific	50	29	6	2.029	0.828
9	tool	106	52	12	2.866	0.827
10	boss	55	13	1	0.764	0.764
11	goodcode	117	28	1	0.761	0.761
12	scrum	41	12	1	0.707	0.707
13	survey	6	3	2	1.000	0.707
14	cost	51	15	1	0.706	0.706
15	selforganisation	31	16	2	0.983	0.695
16	requirements	22	8	1	0.636	0.636
17	design	57	35	6	1.556	0.635
18	management	59	26	2	0.833	0.589
19	xp	69	41	8	1.644	0.581
20	parallel	23	10	1	0.565	0.565
21	general	68	28	2	0.796	0.563
22	book	18	8	1	0.556	0.556
23	standup	27	12	1	0.556	0.556
24	customer	72	52	8	1.511	0.534
25	agile	39	22	4	1.033	0.517
26	group	4	2	1	0.500	0.500
27	support	2	1	1	0.500	0.500
28	sysadmin	2	1	1	0.500	0.500
29	teambuilding	25	14	2	0.700	0.495
30	quality	11	6	1	0.455	0.455
31	humanresources	16	9	1	0.438	0.438
32	organisation	16	9	1	0.438	0.438
33	projectsize	16	9	1	0.438	0.438
34	distributed	14	6	2	0.615	0.435

Continued on next page

rank	keyword	messages	authors	threads	controversiality	w-controversiality
35	iteration	17	10	1	0.412	0.412
36	certification	20	12	1	0.400	0.400
37	personality	5	3	1	0.400	0.400
38	conference	32	16	2	0.516	0.365
39	hardware	11	7	1	0.364	0.364
40	tdd	11	7	1	0.364	0.364
41	citation	9	7	2	0.500	0.354
42	introduction	13	9	2	0.500	0.354
43	refactoring	20	14	2	0.462	0.326
44	listspecific	68	37	6	0.762	0.311
45	room	13	9	1	0.308	0.308
46	stories	10	7	1	0.300	0.300
47	coach	14	10	1	0.286	0.286
48	contact	7	5	1	0.286	0.286
49	teaching	6	4	2	0.400	0.283
50	solo	4	3	1	0.250	0.250
51	codereview	13	10	1	0.231	0.231
52	acceptance	13	10	2	0.300	0.212
53	practice	11	9	1	0.182	0.182
54	career	6	5	1	0.167	0.167
55	optimization	7	6	1	0.143	0.143
56	retrospectives	15	14	3	0.167	0.096
57	personal_development	12	11	1	0.083	0.083
58	article	3	3	1	0.000	0.000
59	evaluation	4	4	1	0.000	0.000
60	interpersonal	2	2	2	0.000	0.000
61	measure	3	3	1	0.000	0.000
62	methodology	2	2	1	0.000	0.000
63	opinion	3	3	1	0.000	0.000
64	research	5	5	2	0.000	0.000
65	roles	5	5	2	0.000	0.000
66	technique	3	3	1	0.000	0.000
67	tracking	2	2	1	0.000	0.000

Table 10.4: Ranking of the "question" subtopics by w-controversiality (XP practices highlighted).

Inside the "question" and the "announcement" categories the subtopics were analyzed. This was done in the same ways as for the main topics using the w-controversiality measurement. The subtopics and their ranking can be seen for the questions in Table 10.4 and for "announcements" in Table 10.5.

rank	keyword	messages	authors	threads	controversiality	w-controversiality
1	website	406	110	24	4.97	1.02
2	article	162	91	24	4.60	0.94
3	testing	59	14	1	0.76	0.76
4	experiment	144	37	1	0.74	0.74
5	proposal	89	23	1	0.74	0.74
6	blog	132	51	11	2.25	0.68
7	survey	60	47	16	2.55	0.64
8	misc	109	37	6	1.14	0.47
9	death	13	7	1	0.46	0.46
10	personally	9	6	1	0.33	0.33
11	talk	43	28	7	0.87	0.33
12	otherthread	16	10	3	0.55	0.31
13	invitation	4	3	1	0.25	0.25
14	study	6	5	2	0.33	0.24
15	position	9	8	7	0.33	0.13
16	quote	8	7	1	0.12	0.12
17	book	9	9	4	0.00	0.00
18	conference	27	27	26	0.00	0.00
19	course	6	6	5	0.00	0.00
20	data	2	2	1	0.00	0.00
21	dojo	4	4	4	0.00	0.00
22	event	8	8	8	0.00	0.00
23	group	2	2	2	0.00	0.00
24	interview	4	4	1	0.00	0.00
25	listspecific	3	3	3	0.00	0.00
26	meeting	25	25	24	0.00	0.00
27	newsletter	2	2	2	0.00	0.00
28	othergroup	4	4	2	0.00	0.00
29	podcast	4	4	2	0.00	0.00
30	poll	2	2	1	0.00	0.00
31	spam	2	2	2	0.00	0.00
32	teaching	3	3	3	0.00	0.00
33	tool	26	26	20	0.00	0.00
34	was	2	2	1	0.00	0.00
35	webinar	3	3	3	0.00	0.00
36	wiki	2	2	1	0.00	0.00

Table 10.5: Ranking of “announcement” subtopics by w-controversiality.

10.3.1 Topic connections

The coding gives an indication about the interconnections between topics. Let us take the code line “question testing refactoring legacycode” as an example. It indicates a connection between testing and refactoring and bewtween refactoring and legacy code. Therefore, the code lines can be used to generate a graph of these

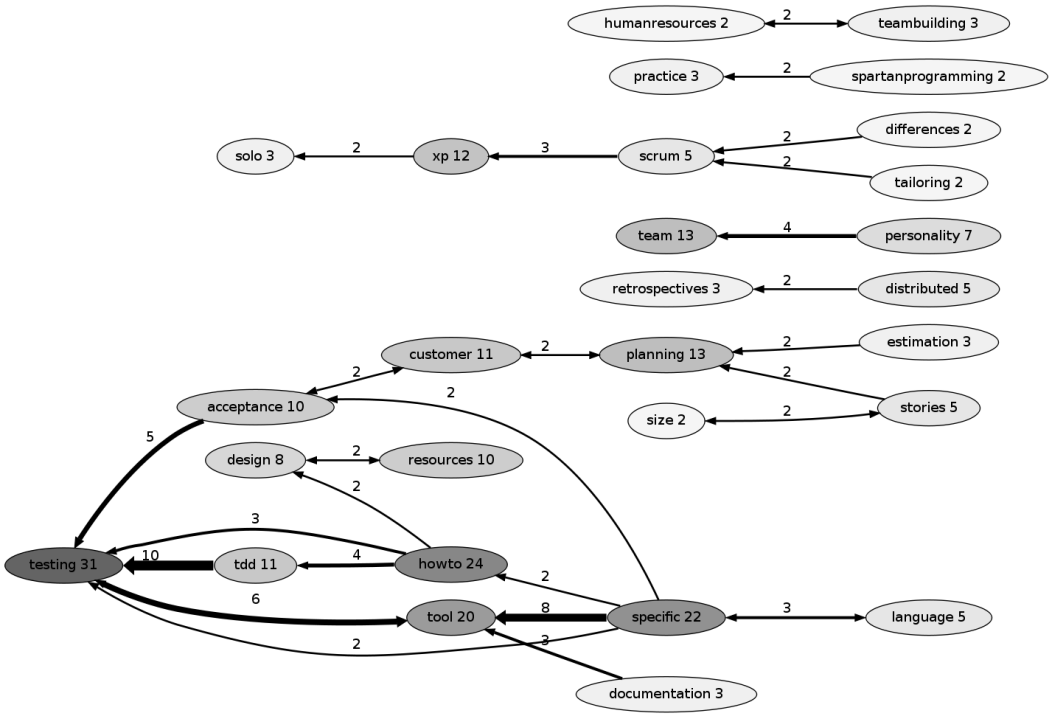


Figure 10.2: Reduced graph of topic connections (edges with strength one removed as well as unconnected nodes)

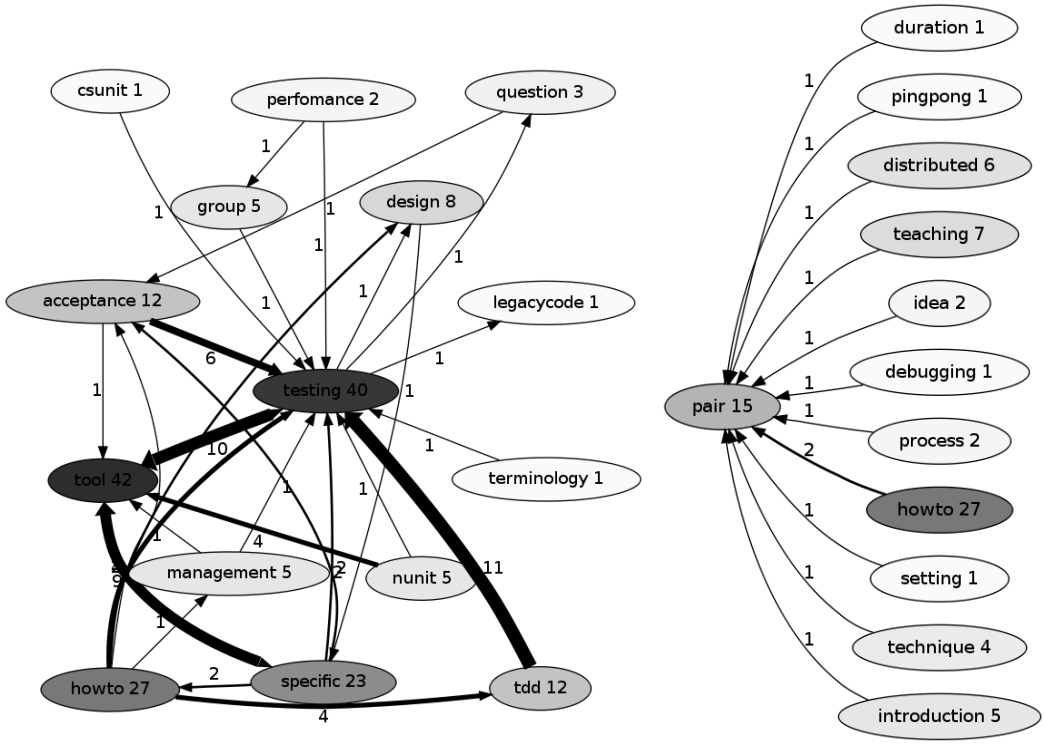


Figure 10.3: Dependency graphs of depth one for “testing” and “pair”.

connections. This graph represents topic dependencies as well as environmental influences to certain practices. The strength of the edges in the graph indicates the strength of the influence (how often this influence was found), while the strength of the node indicates the importance of the topic or how often it was brought up.

For the graph in Figure 10.2 only questions were taken into account. Each topic in the code line represents a node and the connection to the following topic represents an edge. E.g, the example above contains the edges "testing ← refactoring" and "refactoring ← legacycode". For each node and edge the number of occurrences was calculated. Nodes which occur more often were colored darker and edges were drawn thicker. For clarity reasons edges which occurred only once were removed.

In the same way individual practices can be analyzed. This is shown in Figure 10.3, which displays the dependency graphs for the "testing" and the "pair" topic. Here the full set of code lines was used but only the immediate neighbors were taken into account.

The graph representation of the code lines gives a good overview about the underlying structure. It can be seen that the weight of the nodes in this representation does not necessarily correspond to the ranking of topics from Table 10.4. The ranking is based on the w-controversiality measurement and indicates how controversial a topic is, while the dependency graph gives an indication about the influence factors and there strength.

10.4 Results

The analysis was focused mainly on questions, because this was the main goal of the preliminary study. However, also a closer look was taken at the announcements because this category had surprisingly the most initial messages (see Table 10.3).

10.4.1 Questions

Inside the question category about 67 subtopics could be identified. The ranking of these subtopics by w-controversiality shows the popularity of a certain topic. Among these topics some of the XP practices could be identified. Not surprisingly, the most well known practices like testing and pair programming [Sch08] have the highest ranking. This is followed by a request for resources (articles, books, etc.), team related questions, and planning related questions.

Interesting is the "co-location" practice on third place. This topic has only two threads but a high number of messages (168). Most of these messages (154) were contributed by a single thread where the initial post raised the "question" if co-location could double the productivity of agile teams compared to working from home. Here it is the assumption that the combination of the topic productivity as well as the high amount (double) and the homework/company issue together are more responsible for the high discussion potential than the topic itself. It is expected to be an outlier in the distribution.

Another unexpected result was that a practice like the Metaphor was not found. This practice is reported to be not well understood [Wes02] and neglected by many

projects [RS02]. The expectation would be to find it among the questions – which was not the case. So why is the Metaphor practice not mentioned?

Unfortunately the available data is insufficient to answer these questions. To determine the size of the real distribution of the topic "co-location" and to search for hints regarding the Metaphor practice additional data is needed.

10.4.2 Announcements

rank	keyword	threads
1	conference	26
2	website	24
3	article	24
4	meeting	24
5	tool	20
6	survey	16
7	blog	11
8	event	8

rank	keyword	threads
9	talk	7
10	position	7
11	misc	6
12	course	5
13	book	4
14	dojo	4
15	otherthread	3

Table 10.6: Ranking of the first 15 "announcement" subtopics by number of threads.

As was stated above, the announcements are the group with the most initial messages, and therefore deserved a closer look. It can be seen in Table 10.6 that the main purpose of the announcements in the list is – not surprisingly – to announce certain events like conferences, tool releases, etc. These messages have – as expected – almost no responses. The fact that the majority of messages in this group is concerned with real-world events indicates a lively community also outside of the net.

The other large group of messages consists of the announcement of websites, articles, and blogs. As can be seen in Table 10.5, these messages have a high w-controversiality which means that there are discussions going on about these topics. The reason might be that these resources contain mostly some statements or reports about XP which lead to further discussions. Therefore, it seems that they are more related to other groups than to announcements. However, to use this information for analysis, coding of the announced resource will be necessary.

A very interesting fact for researchers is that alone in 2008 16 surveys or studies were announced. This indicates a high scientific interest in agile methodologies and also shows that there should be a lot of data already available for further research.

10.4.3 Topic map

The generated topic map gives an impression of the connections between different topics and their interaction. It is not surprising that in the map in Figure 10.2 the "tdd" topic (Test-Driven Development) and the "tool" topic have strong connections to the "testing" topic. However, it is interesting to note that the "howto" topic – indicating a question for help or guidance – has also a strong connection to "testing" as well as to "pair". It is also interesting that the "pair" topic has a

broad variety of subtopics and no conclusion can be drawn about which topic has the strongest influence. Also here additional data would be needed.

10.5 Conclusion

It could be shown that the XP mailing list can be used as data repository for statistical analysis of the XP context factors and most probably of XP in general. The ranking of the topics found correspond with studies from [Sch08], where pair programming and testing were the most well known agile practices, as well as with our own case study where these two were seen as the most problematic.

The ranking correlates with the expectations derived from case studies and personal experience. As expected, practices which involve personality related issues like Whole team, Pair programming, and On-site customers as well as practices which contradict with the traditional software development approach like Testing, Planning Game, and Simple design are those with the highest ratings. In contrast, practices which are mainly of an organizational or technical level like Stand-up meeting [1] , Small Releases, Continuous Integration, Coding Standards, Collective Code Ownership, and Sustainable Pace have a low ranking or are not mentioned at all.

The Concept map gives an insight into the interconnections of the data. This allows finding influence factors for specific topics or XP practices which can then be analyzed in more detail.

For the preliminary analysis which should mainly evaluate the approach, the result was satisfying. However, to get a better understanding of the XP community a lot more data is needed. Despite the large number of messages evaluated, the average thread count for the topics in Table 10.4 is three – which is poor for a statistical analysis

Also some weaknesses of the analysis process were found. The most obvious weakness is the manual coding which is a huge time effort. For the analysis of the 2008 data it took on average about 2 minutes to read, understand, and code a single message. As the assumption is that there are up to now about 12,000 first messages in the whole list, this would mean 400 hours or about 50 days just for the initial coding.

One way to solve this issue would be to use multiple coders, which would raise the problem of inter-coder reliability [DWSVVK06]. Another way would be to use automated classifiers as suggested in [FRMF07]. Automated classifiers can be used to identify message types, sentiment as well as other properties [FRMF07]. The currently available data can be used to train such classifiers and as a benchmark for evaluating the success of different tools and techniques.

Another problem was the message structure itself. Some messages consisted of more than one concept and included statements, reports, questions, etc. Thus it was sometimes not possible to make a meaningful overall classification of the message and decide the appropriate main topic. A process improvement would

[1] The Stand-up meeting is neither in [Bec99b] nor in [BA04] mentioned as a core practice. Nevertheless it seems to be an important practice in XP and other agile methodologies.

be to choose the individual concepts instead of the whole message as the unit of analysis [DWSVVK06] and code them independently.

During the analysis phase, the codes and the coding structure used, proved also not to be ideal. This was expected, as to search for process improvements was one reason for conducting a preliminary study. During the next step of this research a better coding structure will be used which will comprise a concept-tree instead of one main category and coequal sub categories. It is assumed that with this structure, further information concerning the interdependencies of the topics can be found.

For the upcoming research it is planed to use also the authors for the message classification. Authors seem to be consistent in their opinion [FRMF07] which allows a better analysis of controversiality by detecting how many different opinions are present. Furthermore also the role of the author inside the newsgroup network can be used for classification [FSW06]. It would allow to distinguish between "insider" talks and "new-by" questions, where different topics are expected.

A side effect of this study was the discovery of the large amount of surveys which were announced in this group, which led to the impression that there is much scientific interest in agile software development. Their presence suggests that there should be much data already available to analyze. Also many case study reports are hidden inside the individual messages, which altogether could be used to get a clearer picture of the "real-world" adoption of XP.

Acknowledgements

I want to thank the XP mailing list members for their kind permission to use the list for research purposes.

Chapter 11

How to Introduce an Agile Process

In this chapter I will cover the topic of how to introduces an agile and usability aware development process into an organization. The ideas and suggestions are derived from our own project experiences as well as from the available literature.

11.1 Why Introducing an Agile Process?

A clearly defined development process is inevitable for every SW development company. Therefore, I assume that in most companies there already exists such a process. But is it satisfactory for all involved parties? Does it fit realty? Is it flexible enough to cope with changes? Does it include usability aspects?

In my experience, the honest answer to most of these questions will be NO. There seem to be two kinds of development strategies in practice: Very formal processes and barely defined processes.

Very formal processes mostly exist in bigger organizations. These processes are defined in a clear, strict and logical way. The problem with them usually is that, although they make sense from the logical point of view, they demand actions and tasks which are against the demands of most projects and most developers. For example Cockburn [Coc] mentions a project which failed because the developers had to keep the documentation consistent. Sometimes even the project manager tells the developers NOT to follow certain rules because other things are of higher priority. Although these companies claim to use their defined processes for development, in reality they are rarely followed. Processes like this always bring the developers into a difficult situation: If they follow the process, the product manager will not be happy, otherwise the Quality Assurance (QA) will not be happy. In my last company I brought up the topic that our process was not fitting reality. I got the following answer: “That is the way we live our process”. My translation of this answer is: “We do something different and pass it off as the process because the defined process does not work.”.

Barely defined processes exits in smaller companies. Mostly only the bare minimum of a process is defined – if at all. Development is often done on an ad-hoc basis without defined phases and criteria. This leaves the developers without rule support and the management without tools for predictions.

Both kinds of processes are not desirable in practice. A good development process should fit the reality of SW development and support all involved parities, developers, project manager, QA, management, customer, etc. This is the gap that agile

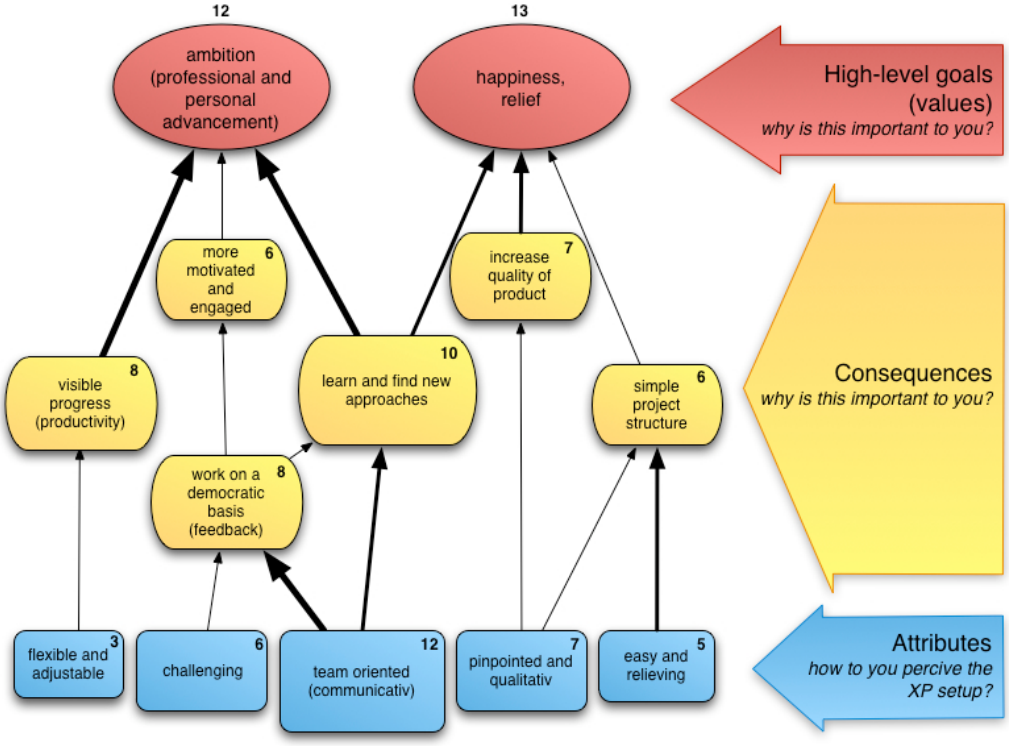


Figure 11.1: Hierarchical value map for XP practitioners [Cur10].

processes aim to fill.

11.2 Proposed Process

Software is developed by people. Therefore, processes should support people and address their needs. Agile processes emphasize this importance of the involved people [Man01]. Our partner CURE conducted a study to find out more about the motivation why people want to apply agile methods. In this study the laddering interview technique was used. The laddering interview technique is a technique based on the means-end theory [RO01]. It allows the determination of the core values which people associate with the properties of a specific product. Figure 11.1 shows the resulting hierarchical value map of Extreme Programming (XP) programmers. The map shows the relevant concepts which XP programmers associate with agile development. The thickness of the arrows indicates the strength of the connection while the numbers indicate the total number of answers in the category [Cur10]. This map gives an indication that people want to use an agile process like XP because they expect higher motivation and job satisfaction as well as products of better quality.

Furthermore, our research has told us that the important aspect of usability is only partly covered by agile processes. If the customer is the real end-user of the system she can influence the development to get a system tailored to her specific needs,

however this is rarely the case. Therefore, an explicit effort has to be made to include usability methods and tools into the process to ensure not only customer but also user satisfaction.

The proposed process tries to take these factors into account. The basis is derived from our research of the XP methodology over the last three years. Additional influences are from the results of literature research as well as my experience with XP and SW development over the last 10 years. The process is iterative and contains elements from scrum as well as from XP. For development, the practices of XP should be used while the organizational part is highly influenced by Scrum.

11.2.1 Main Process Roles

The main roles inside the proposed process are the Product Owner, the Coach, and naturally the developers. I use the Scrum term Product Owner instead of the XP term onsite-customer because in reality there is rarely a real customer on site. Usually, the missing customer is compensated by a so called customer proxy. Therefore, the term Product Owner describes the real world setting more accurately and can also be used for the rare conditions where a real customer is on site.

The Product Owner is the only person responsible for the development concerning the features. She has close contact to the customer and understands her wishes and visions. Her main duty is to bring these visions into the development by acting as an interface between the customer and the team. Additionally, she has to shield the developers from the management.

The Coach is responsible for the process and teaches others the practices and the methods. She supervises the development and enforces the process rules. The Coach should be a senior developer as this position requires authority as well as technical knowledge and development skills.

The developers are responsible for producing the software. They should posses all the skills necessary for creating the product. Additionally, they are required to have high social skills as they are expected to work as a team in close interaction.

11.2.2 Development Cycles

Development is done in an iterative way. The iteration length should be one week in the beginning. There are two reasons for this: First, in the beginning of a project the requirements are usually still pretty unstable and changes occur frequently. Second, if the process is new to the team the shorter periods help to “practice” the process and to reduce the possibility to fall back into old habits. If the process is familiar and the project becomes more stable, longer iteration periods can be chosen. The periods should be small enough that no re-planning during an iteration is required. Similar to Scrum, a once planned iteration should remain stable to give the developers a solid short term goal. Frequent re-planning should be avoided as it reduces the development time, motivation, and trust.

The release length is chosen according to the needs of the project. A release should not be an arbitrary internal deadline. It should be a real deadline where something is delivered. The recommendation is to have frequent and small releases. If this is not

possible, as for example only one big final release is scheduled, artificial intermediate releases are not recommended. I this case the release planning meetings are replaced by regular direction giving meetings. These meetings occur every few iterations and on demand when new developments make them seem appropriate. The purpose of these meetings is keep track and to adjust the high-level view of the project as well as to communicate this vision to the development team. Every team member should have a common high-level view about what should happen in the upcoming months.

11.2.3 Estimation Process

It is very important that the estimation is not done in time as this may lead to confusion and wrong expectations especially from the management side.

An estimate in an abstract unit like story points is recommended. To get a basis for this estimate, the smallest story of the iteration is taken and estimated with two story points. Relative to this story the complexity of other stories can be estimated. Some sources recommend the use of stories of similar size to determine the estimate of a new story [BF00a]. As this approach appears to be more precise it is also more complex. The comparison with older stories takes more time and the results maybe not as precise as anticipated as the impression of the story complexity will be different after the completion. In our project we saw that this pretended precision is rarely necessary and that it is sufficient to estimate based on a feeling than based on concrete comparison.

The estimation process itself is done using planning poker cards. These cards usually contain numbers in the Fibonacci sequence as well as additional symbols[1]. The story is presented by the customer together with the developer who assisted her during the writing process. Then each developer picks one cards from his deck and places it with the number hidden on the desk. After every developer has chosen his card all estimates are shown at once. If the estimates are similar they are used as the final estimate of the story. If some of the results differ, e.g., one developer has a much higher or lower estimate than the others this is discussed as it indicates different views of the story or of the solution. Afterwards the story is estimated again using the described method. This process is repeated until the estimates converge.

This estimation technique has various advantages. Different opinions on stories are revealed fast and only these stories have to be discussed which saves time and energy during planning. The factor of social pressure is reduced by making the estimates individually before showing them to the group. It has also been shown that estimates derived in this way tend to be less optimistic and more accurate [MOH07].

11.2.4 Process Flow

At the start of the project a short requirements gathering phase is done. The main goal of this phase is to get the high-level requirements from the customer and to understand what she wants to accomplish with the product. This is done by the

[1] An example would be cards with the numbers 0, 1, 2, 3, 5, 8, 13, 21, 34, 55 as well as a “?”, indicating the need for clarification.

Product Owner together with the customer. Through this phase the Product Owner is able build a relationship with the customer as well as to generate the first User Stories. The story creation process in this phase is the same as usual including a developer and usability input. Additionally, the customer has to be included as much as possible and confirm these stories. This is crucial as in the initial phase of the project the necessary mutual understanding between customer and Product Owner is still under development. Furthermore, especially these stories concern the core of the system.

The first direction meeting is already held during this phase. It gives the team a high-level view of the project as soon as the basic concepts of the system are clear. This allows the team to prepare itself for the upcoming planning meeting. Therefore, an iteration planning meeting shall not immediately follow a direction meeting. The time to digest the new information would be missing.

The next phase is the iteration planning process. Prior to iteration planning, stories are prepared by the Product Owner together with a developer. The stories should be of a scope that several stories fit into one iteration. I further recommend that the stories are small enough that a task breakdown during planning is not necessary. This usually makes the stories more concrete and avoids discussions during planning. Additionally, the velocity is calculated summing up the estimates of the finished stories of the last iteration. This velocity provides the basis for the iteration planning as it determines the amount of stories which can be scheduled. To take changing preconditions between iterations, like holidays or vacations, into account, the velocity can be scaled using the sum of available person days.

During planning the story is estimated by the developers and prioritized by the customer. The Product Owner brings the stories in a pre-prioritized order to the planning meeting. Then the stories are estimated as described above. After all stories have been estimated in this way the Product Owner makes the final prioritization of the stories. She can fit as many stories into the iteration as the velocity allows. Additionally, a few extra cards are estimated and prepared in case the velocity is exceeded. This avoids the overhead of re-planning and is especially advisable in case of an unstable velocity. If no velocity is available, e.g., at the beginning of a project, then the team guesses how many stories could probably be done.

After the Planning Game, the implementation phase begins. Stories are processed in the order defined by the Product Owner. During their implementation the Product Owner should always be available for clarification. The developers form pairs and each pair picks one story to work on. This story is marked it as being processed. If the pair thinks that a story is finished it presents it immediately to the Product Owner for acceptance. The Product Owner accepts the story and crosses it out on the story board. This physical action by the Product Owner is important as it clearly indicates her decision. Once a story is accepted the pair continues with the next unprocessed story.

A daily stand-up meeting is held during the iteration. This meeting is only for the team. The active participation of the Product Owner is optional. Other persons are allowed to attend but not to talk during the stand-up meeting. The meeting should be short and focused which has to be ensured by the Coach. Each team member tells the team what she has done since the last stand-up, what she intends to do,

and what obstacles she encountered. Problem solving and discussions are not part of this meeting and have to be avoided at any cost. However, these activities can be scheduled during the stand-up. The purpose of this meeting is to synchronize the developers so that everyone knows what is happening and who needs help. Discussions and suggestions take place after the meeting, involving only the team members which are relevant for the respective topics.

At the end of the iteration an iteration review meeting is held[2]. In this meeting the Product Owner and the team talk about the project and the process. The goal is to identify potential problems on both levels and to suggest improvements. These problems are collected together with possible solutions and are prioritized in order of their severity. The process problems are collected in an impediment backlog. This backlog is managed by the Coach who tries to resolve these issues. Project specific problems are collected by the Product Owner as an input for new stories. As agile teams depend heavily on the communication between their members, interpersonal problems are addressed as well. Therefore, it is important that all participants keep the contents of the meeting confidential and that they maintain a climate of honesty and respect. Furthermore, no person from the management is allowed.

11.3 Organizational Challenges

Management support is needed to implement the full process described above. Projects are often conducted in organizations with a matrix structure. This means that the developers have two managers to satisfy: the project manager and the line manager. The line manager is responsible for the development team while the project manager is responsible for the project. Roughly we can say that the project manager determines *what* is developed and the line manager determines *how* the development is done. In the case of agile methods this means that the project manager is responsible for the organizational practices and the line manager is responsible for the development practices. Additionally, the individual developer is responsible for his own development style. Furthermore, the project- as well as the line manager has an upper management to satisfy.

Although there is mutual influence between these responsibilities, this simplified picture allows us to draw some conclusions. We can determine which preconditions have to be fulfilled for the adoption of agile methods on various scales. The project manager is mainly concerned with the organizational part and can usually be seen as the Product Owner. Therefore, the iterative work flow with prioritizations can only be implemented if she supports this way of development. The line manager is responsible for the team and the development of their skills. Therefore, she can enforce the development related practices like Pair Programming (PP), Test-driven Development (TDD), etc. The individual developer alone can only implement a subset of the practices, namely, TDD, Simple design, and Refactoring. In conjunction with the team members the practices of Collective Code Ownership, Coding

[2] A suggestion for a one-week iteration would be to hold the review meeting as well as the planning meeting of the upcoming iteration on the same day, e.g., Monday, as it turned out that at the end of the iteration and week there is rarely time for a meeting. However, in this case an appropriate break, e.g., lunch break, should be scheduled in between.

Standard, and stand-up meeting can additionally be implemented. The degree to what extend each of these people can enforce agile methods depends on the one hand on the others and on the other hand on the support from the upper management.

Support from the customer is desirable but not absolutely necessary. The project manger is the interface to the customer and brings her wishes into the development process. Therefore, a traditional approach can be executed on the outside. However, it is suggested to make the customer aware of the agile nature of the process and to include her as far as possible. The advantages are that there can be closer interaction and a more agile process.

11.4 Personal Challenges

Beside the organizational challenges, personal challenges will be the main problems of an agile methodology in practice. DeMarco and Lister point out the importance of people related issues in SW development [DL99] and Cockburn calls them "the dominant, first-order project driver" [Coc]. After over 20 years of experience he saw a repeated pattern in many projects:

Problem 1: "The people on the projects were not interested in learning our system."

Problem 2: "They were successfully able to ignore us, and were still delivering software, anyway."

He concluded that there are for main characteristics of people which are relevant in SW development:

1. People are communicating beings, doing best face-to-face, in person, with real-time question and answer.
2. People have trouble acting consistently over time.
3. People are highly variable, varying from day to day and place to place.
4. People generally want to be good citizens, are good at looking around, taking initiative, and doing "whatever is needed" to get the project to work.

[Coc]

Different personalities have different needs and goals. Corporate cultures affect the behavior of the employees. Their educational backgrounds as well as prior working experiences have established a set of beliefs and habits. All these factors and more influence the Software (SW) development process. Therefore, the best development process is only as successful as the involved people. From experience I can confirm the statement that: "They were successfully able to ignore us, and were still delivering software, anyway." [Coc].

These challenges appear in any process but some of the agile practices are more likely to encounter problems. The most problematic practices seem to be PP and TDD as both strongly contradict previous experiences. TDD turns the familiar way of developing software upside down. Experienced developers may have to completely relearn their professional basics. PP forces close interaction with co-workers. This reduces personal freedom as well as might magnify personal differences. A great deal of discipline is necessary as the developer is also responsible for his pairing

partner.

To solve these challenges is hard and may be sometimes even impossible. However, the chances can be increased if a few basic principles are taken into account.

Leadership is important. People have to be led by example. They will immediately recognize if their leaders are not committed to their statements. If this is the case they will see no reason for following them. Statements of the leaders have to be backed up by their actions. Only then people can be convinced to follow. This is true for individual leaders as well as for whole corporate cultures [DL99].

Additionally, the leadership style which is necessary depends on the situation. An approach which takes this into account is the situational leadership model [HBJ07]. The model basically states that during different phases of team building different leadership styles are necessary. In a situation where changes are required the leadership should be highly directive and highly supporting. The leader has to tell the team exactly what to do and how to work. This strict leadership provides clear rules and visions about what is expected. Once the change is underway and the rules are established the style gradually should become less strict. The final aim is to create a self-organizing team which requires a delegating leadership style.

It is necessary to generate a motivation for change. To achieve this, three basic components are needed: The current status is perceived as dissatisfying, the goal is clear and desirable, and the road is clearly laid out and seems reasonable. This implies that if the people are content with their current working conditions the opposition against changes will be strong. Even if they see the benefits of the new method they will be reluctant to change. Furthermore, it tells that the closer the goal is and the better the situation gets the less motivation will be left to reach final goal.

People find it hard to change old habits. Rational understanding and agreement is a precondition but unfortunately not sufficient. Even if people want to change some aspects of their work they may get lax over time and finally fall back into old habits. Therefore, self-discipline is required and constant attention has to be paid to avoid this. To support this process an experienced Coach is recommended.

Common experiences are the basis for mutual understanding. If a new method is introduced the team members should get a common vision of it. This can be achieved by sending the whole team together to training. The common experiences during this training will give the team a common base and reduces friction.

11.5 Usability Challenges

The usability inclusion into a SW development process is an important issue. In the process described above there are three different phases of usability input. Each of

these phases has a different characteristic and needs a different form of input. Additionally, a constant source of usability input can be created by using the Personas method.

During the creation and preparation of User Stories the usability input should ensure that the story is created in a usability aware way. If Personas are present their names should be used on the story cards. The usability instruments which can be used in this phase are heuristics and expert evaluations. As this is the preparation for an upcoming planning meeting, usually a few stories will be evaluated at once. In this phase, whole interfaces are designed and all usability related aspects of the stories are discussed. Therefore, this could take some time and close interaction with the Usability Engineer (UE) is recommended. The best option is face-to-face communication followed by video conferencing and phone calls. Emails are not recommended with the exception of distributing interface sketches as base for discussions.

During the implementation of the stories ad-hoc input can be needed to clarify some smaller points or to discuss necessary changes. If Personas are present their names should be used during these discussions. The usability methods are similar to the ones described for the story writing process. However, this input is more pressing but is usually limited to small details and single stories. The timing here is more crucial but the scope of the requested input is smaller. Therefore, if the UE is not on site, phone calls and even emails may work well.

After the implementation of the story the resulting features can be tested. This will usually be done at the end of an iteration. The minimal scope of these tests will be all recently implemented stories. Usability methods which can be used are expert evaluations, cognitive walkthroughs, and user tests. These tests serve as input for the story creation process. The effort for these tests is higher but the reporting period can be longer. However, depending on the stability of the project care must be taken that the results are still valid at reporting time. Not too much interface changes should have occurred in the meantime.

Beside the UE the whole team should be trained in usability methods to encourage user centered thinking. The project relevant usability heuristics should be known by everyone. This is especially true for the Product Owner as she is the one who creates the User Stories. In this way also a project which has no access to a dedicated UE can at least partly compensate this shortcoming.

11.6 Adopting the Agile Process

Introducing and adopting an agile process in an organization can be done on several levels and to various degrees. The two main levels are the organizational level and the development level.

The popularity of Scrum indicates that adoptions on the organizational level are easier to make. Management is used to define processes and hold meetings. In every project similar meetings are held, prioritizations are made, and project plans are developed. Only the time-frame is much smaller in agile projects. So actually, no real change in behavior is required. However, the active support and commitment

of the upper management is crucial for a successful adoption.

On the development level this support is also needed, but other problems occur which may require special attention. The main reason for these problems is that on this level greater changes in the daily working routine are required than on the organizational level. To overcome these problems and to increase the chance of a successful adoption there are some tools which can be used.

11.6.1 Pair Programming as Catalyst

There are many reasons why PP is an important practice. PP increases the knowledge transfer, reduces the truck factor, and there is evidence that it increases job satisfaction [SPMW02]. A study at the University of Utah showed that pairs are 15% slower than two individual programmers [Wil00] but they also produce 15% fewer defects. Taking into account the time and costs to fix errors, the overall performance of a pair of programmers is better than the combined effort of two individual programmers.

But especially in the case of introducing a new process, introducing a new team member, or building a whole new team, pair programming can act as a catalyst. Because there are two factors which play important roles in these processes: learning and social interaction.

Studies have shown that pair programming is of significant help in learning situations [NWW+03]. The situation resembles the constructivists' learning theory [Wik05] where internalized concepts, rules, and general principles are applied in a practical real-world context.

According to Jerome Bruner and other constructivists, the teacher encourages students to discover principles for themselves in a hands-on approach and to construct knowledge by working to solve realistic problems, usually in collaboration with others. This collaboration is also known as "knowledge construction as a social process". Some benefits of this social process are:

1. Students can work to clarify and organize their ideas so they can voice them to others.
2. It gives them opportunities to elaborate on what they have learned.
3. They are exposed to the views of others.
4. It enables them to discover flaws and inconsistencies.

	With Pairing	Without Pairing
Assimilation (a) (total days)	12	27
Mentoring (m)(% of day)	26%	37%

Table 11.1: Survey values for assimilation and mentoring.

Studies confirm this assumption also in the field of programming [WK00]. A survey in 2001 investigated the effect of pair programming on adding new team members to an existing project [WSA04]. Table 11.1 shows the resulting data which indicates that pairing halves the training period.

Social interaction is closely coupled with learning and team performance [The95]. Highly, self-organizing teams need to touch the emotional level of the relationship between the team members [LT01, Stu05]. To find and solve conflicts is a very important step on the way to a successful team. PP is a highly effective tool to accomplish this. The close interaction of the partners during PP speeds up the development on the emotional level. It acts as a catalyst for positive as well as negative developments. The team members get quicker accustomed to each other but also conflicts are more likely to break out.

As constant pairing can become emotional exhaustive possibilities for retreat have to be provided. The problem can be solved by the means of spike development. A spike is a small program to explore a certain technology which is likely to be used later in the project. This is done solo and the gained experience is later incorporated into the project. In this way the programmers can have some freedom and solo time.

11.6.2 Coding Dojos to foster Development Practices

Development work should be done in pairs using the practices defined by XP. The most important practices are PP, TDD, YAGNI, and the daily stand-up meeting. But these practices are also the hardest to implement. Therefore, tools are needed which help in introducing these development practices.

The introduction of new methods needs time and a safe environment. Under the pressure of an ongoing project people will quickly fall back to their old habits [WvD07]. Therefore, the best situation to introduce new agile development practices would be a pilot project which is mainly for training purposes. As this is rarely possible other methods have to be applied.

XP practitioners have invented an alternative called “Coding Dojo”. The term “Dojo” (道場) comes from Japanese martial arts. A Dojo is a training place, a place (場) where you learn the way (道) of an art. Therefore, a Coding Dojo is a place where developers learn and practice how to develop.

Usually developers learn on the job which is always a difficult situation. A Coding Dojo gives the developers the freedom to learn without pressure and make mistakes without consequences. Therefore, the task presented during practice should not necessarily correspond with the normal work. Otherwise the element of freedom and fun might be reduced which affects the positive effects. A good idea would be to develop something the whole team is interested in like a small game, a new tool, etc. This will increase the fun factor and therefore the motivation for actively participating in the Dojo.

A Coding Dojo is also a good team experience. Coding Dojos should be held regularly during working time. The intervals may vary depending on the maturity of the team and the process. During the introduction of new practices the Dojos should be held more frequently. I recommend holding a small Coding Dojo of one to two hours duration at the beginning of every iteration. In the process described above, a good time would be before the Planning Game as afterwards the new tasks at hand will occupy the developers. In later stages, a monthly Dojo might be sufficient. However, this highly depends on the attitudes and preferences of the developers and the progress of the process.

This might seem as a waste of development time but I strongly recommend it as a highly effective measure for team-building. The motivation and gained experience will in term lead to higher quality and productivity.

Coding Dojos enforce development related practices like PP, TDD, Simple Design, Refactoring, Collective code Ownership, and Coding Standards. A few of the main techniques developed for this purpose are described below.

Kata Programming: The term "Kata" (型) also comes from the Japanese martial arts and means model, template, or pattern. A Kata in martial arts is a fixed set of movements. The idea behind is to get familiar with a predefined set of attack and defense movements. This set is practiced over and over again and gives the student some basic tools to cope with different situations. A similar approach can be used for learning programming skills.

An XP Kata is a fixed way to implement some small application. By executing the steps of the Kata over and over again the developer experiences and internalizes this approach to development. The Kata can be seen as a series of useful patterns connected together which give the developer a toolbox how to handle similar situations. She gets a feeling how TDD can be applied, how small the individual steps can be made, how the simplicity principle can be applied, how Refactoring is done, etc.

"One of the most typical ways in which people learn is by inferring general rules from examples" [HR77] A Kata is an example how to solve different situations. Practicing a programming Kata is well suited to get developers going with new practices as they are given patterns to solve standard situations.

Randori Programming: The term Randori (乱取) comes from the Japanese martial arts[3]. It describes a partner exercise where, contrary to the Kata, the attacks and defenses are not fixed. This is an advanced form of practice where the principles learned by the Kata are applied to different practical situations.

In the XP community this term represents a free coding exercise of a pair in front of an audience. During the Randori some small application is developed. A Coding Randori usually consists of one pair developing while the code is projected with a beamer and an audience which is watching. The pair works for a certain time, e.g., 5 minutes. Then the driver goes into the audience, the navigator becomes the driver, and one member of the audience becomes the navigator. Goal of each pair is to develop in simple micro steps and to accomplish as many red-green-refactoring cycles as possible in the given time.

A Randori allows the audience to see different approaches. The constant switching of pair partners ensures that the audience keeps its attention. Furthermore, it forces the people in the audience to think in the paths the current pair is developing in and continue this work. This practice allows teaching many people at the same time and encourages pairing.

[3]Especially from Judo.

Ping-Pong Programming: The basic idea of Ping-Pong programming is that one developer writes the test while the other implements the functionality. This is done in small steps with frequent switching. Contrary to the practices above, this practice can and should be readily applied outside the Dojo during normal development.

This method is ideally suited for pairing with novices, either in XP or in the team. The experienced developer who writes the tests teaches the novice during the session how TDD is performed and how the system works. At the same time she determines the design of the implementation. The novice is implementing this design and gets feedback by making the tests pass.

Important here is that this tool is used consciously and the roles are clearly defined. Then the experienced developer will have the patience to guide the novice.

11.7 Conclusion

Agile development methods have many advantages. They increase customer satisfaction, have a higher probability of project success, motivate the developers, etc. These are compelling reasons to use agile methods in practice. However, the inclusion of usability methods is usually missing.

Therefore, a process is proposed which covers all these aspect. The proposed process is derived from our experience with the XP methodology which was used by our team for three years. Basic concepts have been adopted and modifications were made to address detected shortcomings. Our experience with the inclusion of usability methods is incorporated as well as additional suggestions form literature research.

Also ways for the successful adoption of this process are shown and potential problems are addressed. Different levels of adoption as well as the possibilities and influences of key-players are mentioned. Problem domains are examined and factors which influence the adoption are mentioned. Furthermore, tools and strategies to address these problems are given.

Chapter 12

Summary

The goal of our project was the examination of Extreme Programming (XP) in a real world context. Additionally, the usability inclusion into agile methods was examined. Therefore, our team developed a multimedia streaming application for mobile phones and laid special focus on agile software development methodologies and usability.

This type of application was chosen because the current trend in mobile multimedia consumption is more than obvious: it is becoming more and more popular. According to current trends, the community of mobile phone application users will grow rapidly and today's consumers are willing to pay a reasonable price for this service [CW07]. Therefore, we had partners from industry which tried to sell our product. This provided a real business background for the research which resembled real world conditions.

The most critical factor for this kind of application is user acceptance. To address this issue, in our software engineering process, usability engineers were accompanying the development team during the whole project life cycle. While developing this application we took all efforts to satisfy the needs of the end user. We combined XP with User Centered Design (UCD), integrated HCI instruments in our development process, developed an iterative UI development process, conducted usability studies and did studies on the relation of basic human values to behavior patterns of the usage and production of mobile multimedia content.

In our development process, the Usability Engineers (UEs) provided suggestions which were incorporated continuously into the system. This process was facilitated by the short development iterations and has proved to provide early and valuable feedback. The test-driven development approach allowed converting these findings into a set of automated tests. These tests defined the functionality of the system, served as specifications for the development, and prevented previously discovered usability problems from reappearing. Furthermore, the inclusion of test-users provided additional benefits. Developers who watched these tests became more usability aware, new User Stories could be created based on the observations, and the development direction could be adjusted towards the wishes of the users.

Additionally, the development process itself was investigated. It turned out that the adoption of the XP practices was much more complicated than the implementation of the usability methods. Problems occurred especially with personality and experience related practices like Pair Programming (PP) and Test-driven Development (TDD). The different backgrounds and educations of our developers together with their distinct personalities caused many conflicts. Indications for the

problems were found by analyzing our data and reasons for the problems were determined. To solve some of these issues a few strategies were successfully applied. Other solutions are suggested and a new usability aware agile process is described. Additionally, tools for the introduction are mentioned.

Usability Results

Through this process we have gathered some experience in the combination of XP and Human Computer Interaction (HCI) centered methods. The actual state of this experience is summarized in this list:

- To understand the basics and importance of usability a short usability workshop should be held for the whole team at the start of the project.
- A highlight video of usability tests should be created and should be shown to the whole team. Highlight videos save time compared to the full video documentation of usability tests and support a user-centered mind-set.
- Personas should be created and actively used, e.g., during the story creation process.
- As the customer is the one creating and prioritizing the stories it is her duty to include usability aspects in the User Stories.
- Usability clarification of stories prior to the planning process is particularly important. Therefore, the customer should consult the UE during the story writing process.
- Proper Customer and HCI engineer coordination is necessary for the inclusion of the usability process in the development. An experienced on-site XP customer can fill in the technical gap between HCI engineers and the developers.
- UEs should be trained in XP-story writing to be able to deliver their findings in the form of User Stories. This saves an additional step and makes them immediately usable during the Planning Game.
- Personal visits of the UE are preferable but communication with video conferences, phone calls, or emails is possible. However, attention hast to be paid to use the right medium for the task at hand.
- The UE should be readily available for short questions during the implementation process.
- Rough, hand-drawn paper mock-ups are usually sufficient.
- If high-fidelity mock-ups are needed they should be implemented into the application. This shows the real look and is reusable.
- The reporting periods are crucial and should be adapted to the process. Acceptable delays are days prior to planning, hours during implementation, and weeks after implementation.
- Usability unit test should be created if possible. However, current frameworks might not be sufficient.

XP Results

During our application of XP several strategies were applied and several problems were discovered. The following short list describes these issues:

- To provide more concrete stories a developer should write them together with the customer.
- Stories should be small enough that no task breakdown is needed.
- Stories should be estimated using the planning poker technique. This speeds up the planning process as discussions are minimized.
- Estimates should not be in time as this might lead to confusion.
- Iterations should be kept stable. Replanning should be avoided as much as possible.
- The Iteration length should be set according to the stability of the project and the experience of the team. Unstable projects as well as inexperienced teams need shorter iterations.
- TDD has to be explicitly learned as most developers are not used to it.
- An able coach is an absolute necessity, especially during an adoption phase.
- Leadership plays an important role and different leadership styles are necessary during a project.
- A training of the whole team together is recommended because this provides a common vision of the process.
- Regular small training sessions in a Coding Dojo should be held to avoid the slow decay of the method.
- Stand-up meetings are small scale planning meetings which synchronize the team. They should be held focused and discussions have to be avoided at any cost.
- An Informative Workspace needs both, PP and stand-up meetings. PP tells *how* work is done while stand-up meetings tell *what* work is done.
- Apply the Simplicity and YAGNI principles as much as possible. Not only the development but also the discussion culture will profit.
- Human aspects are much more important and critical than technical ones. Software (SW) development depends on people and the people on the project might not be interested in learning a new system [Coc].
- Technical practices like Continuous Integration (CI) are not problematic.
- Organizational practices like Planning Game or Stand-up meeting might be problematic if the interaction between the developers is not well established.
- Personality related activities like PP and discussions can be very problematic.
- Communication is important but long discussions reduce the motivation of a team.
- Practices like TDD which goes against previous experiences and habits are very problematic.
- PP acts as a catalyst for team building or team destruction as the close

interaction enforces developments on the emotional level.

12.1 Future Research Directions

For the future I see different directions of research. One is Automated Usability Evaluation (AUE) and its integration into software development processes. The need for better AUE is obvious: the exponentially growing number of custom software products is not accomplished with a similar growing number of HCI experts. Research on testing-frameworks for AUE will be an important next step where HCI experts and developers should collaborate to ensure the final frameworks fulfill requirements of both disciplines. Currently, unit testing frameworks do not support usability testing and specialized AUE applications can not easily be included into an automated build process. Hence, tools will be needed which bridge this gap.

A second research direction is the role of humans in the development process. What expectations and believes do developers associate with certain development processes? Do different personality types have different preferences for development processes? What are the most important personality characteristics of an agile developer? How is the decision making process executed in SW development? What is the ideal team composition for an agile team? To what extend do teams and individual developers really follow their development method? What motivates developers to adopt a practice and what prevents the adoption? These are only some of the questions which are interesting in this context. Obviously, answers to these questions could improve the understanding of some important issues. This might lead to the development of new training techniques which in term could improve SW quality as well as developer satisfaction.

Appendix

Glossary

AUE Automated Usability Evaluation
AV Audio and Video
CC Collective Code Ownership
CI Continuous Integration
HCI Human Computer Interaction
ICT Information and Communication Technologies
PC Personal Computer
PP Pair Programming
QA Quality Assurance
SLOC Source Lines of Code
SWE Software engineering
SW Software
TDD Test-driven Development
UCD User Centered Design
UE Usability Engineer
UI User Interface
XP Extreme Programming

Bibliography

[Abd04] Maysoon Abdulkhair. *A Multilingual Automated Web Usability Evaluation Agent*. PhD thesis, University of Sheffield, England, 2004.

[AD00] Henk Aarts and Ap Dijksterhuis. The automatic activation of goal-directed behaviour : The case of travel habit. *Journal of Environmental Psychology*, 20(1):75–82, March 2000.

[Ale01] F.M. Alexander. *The Use of the Self*. Orion Publishing, 11 2001.

[All08] Business Software Alliance. Software industry facts and figures. Technical report, Business Software Alliance, 208.

[amb] http://www.ambysoft.com/surveys/success2007.html. last visited: 15 Jan 2008.

[Amb08] Scott W. Ambler. *Tailoring Usability into Agile Software Development Projects*. Springer London, 2008.

[App] Apple. Open source streaming server. http://developer.apple.com/-opensource/server/streaming/index.html. Last visit: 28.08.2007.

[AS07] Richard Atterer and Albrecht Schmidt. Tracking the interaction of users with ajax applications for usability testing. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1347–1350, New York, NY, USA, 2007. ACM.

[AWSR03] Pekka Abrahamsson, Juhani Warsta, Mikko T. Siponen, and Jussi Ronkainen. New directions on agile methods: a comparative analysis. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 244–254, Washington, DC, USA, 2003. IEEE Computer Society.

[BA04] Kent Beck and Cynthia Andres. *Extreme Programming Explained : Embrace Change (2nd Edition)*. Addison-Wesley Professional, November 2004.

[Bac99] James Bach. What software reality is really about. *Computer*, 32(12):148–149, 1999.

[BBC] BBC. Online video 'eroding TV viewing'. http://news.bbc.co.uk/2/hi/entertainment/6168950.stm. Last visit: 31.05.2007.

[BCS93] Sandrine Balbo, Joelle Coutaz, and Daniel Salber. Towards Automatic Evaluation of Multimodal User Interfaces. In *IUI '93: Proceedings of the 1st International Conference on Intelligent User Interfaces*, pages 201–208, New York, NY, USA, 1993. ACM Press.

[Bec99a] Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[Bec99b] Kent Beck. *Extreme Programming Explained: Embrace Change (1st Edition)*. Addison-Wesley Professional, 1999.

[BF00a] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[BF00b] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[BFB07] Raquel Benbunan-Fich and Alberto Benbunan. Understanding user behavior with new mobile applications. *Journal of Strategic Information Systtems*, 16(4):393–412, 2007.

[BGD+06] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143, New York, NY, USA, 2006. ACM.

[BJK09] Michael Budwig, Soojin Jeong, and Kuldeep Kelkar. When user experience met agile: a case study. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems CHI*, pages 3075–3084. ACM, 2009.

[Bli] Blinkx. Video Search Engine. http://www.blinkx.com. Last visit: 01.11.2007.

[Bro95] Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Addison-Wesley Pub Co, 2 nd edition, 1995.

[Bro05] G. Broza. Early community building: a critical success factor for xp projects. *Agile Conference, 2005. Proceedings*, pages 167–172, July 2005.

[BRWH00] Gerhard Backfried, Jürgen Riedler, Alfred Walter, and Ken Hampel. Multimedia archiving with real-time speech and language technologies. 2000.

[cha03a] Chaos chronicles v3.0. Technical report, The Standish Group, 2003.

[Cha03b] Jason Charvat. *Project Management Methodologies: Selecting, Implementing, and Supporting Methodologies and Processes for Projects (Youth communicates)*. Wiley, 2 2003.

[CK09] Peter Wolkerstorfer Christina Köffel. Fokusgruppe: Mamtam field trial mit schwerpunkt „mobile multimedia“. Technical report, CURE - Center for Usability Research and Engineering, 2009.

[CL99] Larry L. Constantine and Lucy A. D. Lockwood. *Software for use: A practical guide to the models and methods of usage-centered design*. ACM Press/Addison-Wesley, 1999.

[CL03] Larry L. Constantine and Lucy A. D. Lockwood. Usage-centered

software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice. In *ICSE '03*, pages 746–747. IEEE Computer Society, 2003.

[Coc] A. Cockburn. Characterizing People as Non-Linear, First-Order Components in Software Development, Humans and Technology Technical Report. Technical report, TR 99.05, Oct. 1999, 7691 Dell Rd., Salt Lake City, UT 84121 USA.

[Coo99] Alan Cooper. *The Inmates Are Running the Asylum.* Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1999.

[Coo04] Alan Cooper. *The Inmates Are Running the Asylum : Why High Tech Products Drive Us Crazy and How to Restore the Sanity (2nd Edition).* Sams, February 2004.

[CSM06] Stephanie Chamberlain, Helen Sharp, and Neil Maiden. Towards a framework for integrating agile development and user-centred design. In *7th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2006*, volume 4044 of *LNCS*, pages 143–153, Heidelberg, Germany, 2006. Springer Verlag.

[cur] A field trial case study: Context & experience of a mobile video on demand service over time.

[Cur10] Cure. Enhancing user-centredness in agile teams: A study on programmer's values for a better understanding on how to position usability methods in xp. Technical report, CURE - Center for Usability Research and Engineering, 2010.

[CW01] Alistair Cockburn and Laurie Williams. The costs and benefits of pair programming. pages 223–243, 2001.

[CW07] Christer Carlsson and Pirkko Walden. Mobile TV - to live or die by content. *Proceedings of the 40th Hawaii International Conference on System Sciences - 2007*, 1:12, 2007.

[Dat] DataMonitor. Software: Global industry guide. http://www.infoedge.com/product_type.asp?product=DO-4959. visited: 12.03.2010.

[DL99] Tom Demarco and Timothy Lister. *Peopleware : Productive Projects and Teams, 2nd Ed.* Dorset House Publishing Company, Incorporated, 2nd edition, February 1999.

[DLCA06] R. Dorat, Matthieu Latapy, B. Conein, and N. Auray. Multi-level analysis of an interaction network between individuals in a mailing-list. *liafa.jussieu.fr*, pages 320–344, 2006.

[DLH06] Jen-Wen Ding, Chin-Tsai Lin, and Kai-Hsiang Huang. ARS: An adaptive reception scheme for handheld devices supporting mobile video streaming services. In *International Conference on Consumer Electronics. ICCE '06*, volume 1, pages 141– 142, 2006.

[DR08] Amit Deshpande and Dirk Riehle. The total growth of open source, 2008.

[DWSVVK06] B. De Wever, T. Schellens, M. Valcke, and H. Van Keer. Content analysis schemes to analyze transcripts of online asynchronous discussion groups: a review. *Comput. Educ.*, 46(1):6–28, 2006.

[emm] Emma: Java code coverage tool. http://emma.sourceforge.net/.

[eve] Everyzing. http://www.everyzing.com. Last visit: 01.11.2007.

[Fau03] Laura Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 2003.

[FC59] Leon. Festinger and J. M. Carlsmith. Cognitive consequences of forced compliance. *J Abnorm Psychol*, 58(2):203–210, March 1959.

[Fes97] Leon Festinger. *A Theory of Cognitive Dissonance [1957]*. Stanford University Press, Stanford, 1997.

[FNB07a] Jennifer Ferreira, James Noble, and Robert Biddle. Agile development iterations and UI design. In *Agile 2007*, pages 50–58. IEEE Computer Society, 2007.

[FNB07b] Jennifer Ferreira, James Noble, and Robert Biddle. Up-front interaction design in agile development. In *8th International Conference on Agile Processes in Software Engineering and eXtreme Programming, XP 2007, Jun 18-22 2007*, volume 4536 NCS, pages 9–16, Heidelberg, D-69121, Germany, 2007. Springer Verlag.

[Fow99] Martin Fowler. *Refactoring - Improving the design of existing code*. Adison Wesley, 1999.

[FPB87] Jr. Frederick P. Brooks. No silver bullet: essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

[FRMF07] Blaz Fortuna, Eduarda Mendes Rodrigues, and Natasa Milic-Frayling. Improving the classification of newsgroup messages through social network analysis. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 877–880, New York, NY, USA, 2007. ACM.

[FSM08] D. Fox, J. Sillito, and F. Maurer. Agile methods and User-Centered design: How these two methodologies are being successfully integrated in industry. In *Agile, 2008. AGILE '08. Conference*, pages 63–72, 2008.

[FSW06] Danyel Fisher, Marc Smith, and Howard T. Welser. You are who you talk to: Detecting roles in usenet newsgroups. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 59.2, Washington, DC, USA, 2006. IEEE Computer Society.

[GBRM+09] Jesus M. Gonzalez-Barahona, Gregorio Robles, Martin Michlmayr, Juan JosÃ© Amor, and Daniel M. German. Macro-level software evolution: a case study of a large software compilation. *Empirical Software Engineering*, 14(3):262–285, June 2009.

[GGB03a] Bengt Göransson, Jan Gulliksen, and Inger Boivie. The usability design process - integrating user-centered systems design in the software development process. *Software Process: Improvement and Practice*, 8(2):111–131, 2003.

[GGB+03b] Jan Gulliksen, Bengt Göransson, Inger Boivie, Stefan Blomkvist, Jenny Persson, and Äsa Cajander. Key principles for user-centred systems design. *Behaviour & Information Technology, Special Section on Designing IT for Healthy Work.*, Vol. 22 No. 6:397–409, 2003.

[GH01] Robert Gittins and Sian Hope. A study of human solutions in extreme programming. In *In G. Kadoda (Ed). Proc. PPIG 13*, pages 41–51, 2001.

[Goo] Google. About gmail. http://mail.google.com/mail/help/intl/en/-about.html. Last visit: 25.05.2007.

[Gre05] Alan Green. How to estimate a software project. http://cardboard.nu/docs/howto_estimate.html, 05 2005.

[HAL] Rob W. Holland, Henk Aarts, and Daan Langendam. Breaking and creating habits on the working floor: A field-experiment on the power of implementation intentions. *Journal of Experimental Social Psychology*, In Press, Corrected Proof.

[Har97] Harvard Business School. *Project Management Manual*, 1997.

[HBJ07] Paul H Hersey, Kenneth H Blanchard, and Dewey E Johnson. *Management of Organizational Behavior (9th Edition)*. Prentice Hall, 9 edition, 9 2007.

[HBK] Marc Hassenzahl, Michael Burmester, and Franz Koller. AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität. In G. Szwillus and J. Ziegler, editors, *Mensch and Computer 2003: Interaktion in Bewegung*, pages 187–196, Stuttgart. B. G. Teubner.

[HES+05] Andreas Holzinger, Maximilian Errath, Gig Searle, Bettina Thurnher, and Wolfgang Slany. From extreme programming and usability engineering to extreme usability in software engineering education (XP+UE→XU). In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 2*, pages 169–172, Washington, DC, USA, 2005. IEEE Computer Society.

[HHH92] Monty L. Hammontree, Jeffrey J. Hendrickson, and Billy W. Hensley. Integrated data capture and analysis tools for research and testing on graphical user interfaces. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 431–432, New York, NY, USA, 1992. ACM.

[HL01] Jason I. Hong and James A. Landay. WebQuilt: A Framework for Capturing and Visualizing the Web Experience. In *WWW '01: Proceedings of the 10th International Conference on World Wide*

Web, pages 717–724, New York, NY, USA, 2001. ACM Press.

[HLM+] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Integrating Extreme Programming and User-Centered Design. In *PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference, Lancaster University, UK. 10th - 12th September 2008.*

[HLM+07a] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Thomas Vlk. The sustainable application of leadership and teaming in hrm - models for the today's organisations' needs – a meta-level perspective. 10 2007.

[HLM+07b] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, Thomas Vlk, and Peter Wolkerstorfer. User interface design for a content-aware mobile multimedia application: An iterative approach. In *Frontiers in Mobile and Web Computing: Proceedings of MoMM2007 & iiWAS2007 Workshops*, volume 231, pages 115–120, Jakarta, Indonesia, December 2007. Second International Workshop on Mobile Multimedia Information Retrieval (MoMIR), 3-5 December, 2007, Jakarta, Indonesia ISBN : 978-3-85403-231-1.

[HLM+08a] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Thomas Vlk. Optimizing Extreme Programming. In *ICCCE 2008: Proceedings of the International Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia*, pages 1052–1056. IEEE, 10 2008. The 16th International Conference on Computers in Education, ICCE 2008, October 27-31, 2008.

[HLM+08b] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, Thomas Vlk, and Peter Wolkerstorfer. User interface design for a mobile multimedia application: An iterative approach. 02 2008. Published 1st International Conference on Advances in Computer-Human Interaction (ACHI 2008) February 10-15, 2008 - Sainte Luce, Martinique http://www.iaria.org/conferences2008/ACHI08.html The ACHI 2008 Proceedings will be published by IEEE Computer Society Press and on-line via IEEE XPlore Digital Library. IEEE will index the papers with major indexes.

[HLM+08c] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Agile User-Centered Design Applied to a Mobile Multimedia Streaming Application. In *USAB 2008*, volume 5298/2008 of *Lecture Notes in Computer Science*, pages 313–330. Springer Berlin / Heidelberg, November 2008. USAB 2008 - Usability & HCI for Education and Work, Graz / Austria, 20-21 November 2008 (http://usab-symposium.tugraz.at/).

[HLM+09] Zahid Hussain, Martin Lechner, Harald Milchrahm, Sara Shahzad,

Wolfgang Slany, Martin Umgeher, and Peter Wolkerstorfer. Concept and design of a contextual mobile multimedia content usability study. volume 0, pages 277–282, Los Alamitos, CA, USA, 02 2009. IEEE Computer Society. The Second International Conferences on Advances in Computer-Human Interactions. ACHI 2009. February 1-7, 2009 - Cancun, Mexico.

[HLSS08] Zahid Hussain, Martin Lechner, Sara Shahzad, and Wolfgang Slany. Inside view of an extreme process. In *Agile Processes in Software Engineering and Extreme Programming*, LNBIP, pages 226–227. Springer Verlag, 06 2008. 9th International Conference, XP 2008, Limerick, Ireland, June 10-14, 2008.

[HMS+09] Zahid Hussain, Harald Milchrahm, Sara Shahzad, Wolfgang Slany, Manfred Tscheligi, and Peter Wolkerstorfer. Integration of extreme programming and user-centered design: Lessons learned. In Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *XP 2009*, volume 31 of *LNBIP*, pages 174–179. Springer, 2009.

[HN07] Tasha Hollingsed and David G. Novick. Usability inspection methods after 15 years of research and practice. In *SIGDOC '07: Proceedings of the 25th annual ACM international conference on Design of communication*, pages 249–255, New York, NY, USA, 2007. ACM.

[Hol05] Andreas Holzinger. Usability Engineering for Software Developers. *Communications of the ACM*, 48(1):71–74, January 2005.

[HR77] F. Hayes-Roth. Learning By Example. 1977.

[HS06] Andreas Holzinger and Wolfgang Slany. (XP+UE→XU) praktische erfahrungen mit extreme usability. *Informatik Spektrum*, 29(2):91–97, 2006.

[HSN07] Andreas Holzinger, Gig Searle, and Alexander K. Nischelwitzer. On some aspects of improving mobile applications for the elderly. In Constantine Stephanidis, editor, *HCII Universal Access in HCI*, volume 4554 of *Lecture Notes in Computer Science*, pages 923–932. Springer, 2007.

[HT04] Orit Hazzan and James E. Tomayko. Human aspects of software engineering: The case of extreme programming. In *XP*, pages 303–311, 2004.

[Hud05] William Hudson. A tale of two tutorials: a cognitive approach to interactive system design and interaction design meets agility. *interactions*, 12(1):49–51, 2005.

[IH00] Melody Y. Ivory and Marti A. Hearst. The state of the art in automated usability evaluation of user interfaces. Technical Report UCB/CSD-00-1105, EECS Department, University of California, Berkeley, Jun 2000.

[IH02] Melody Y. Ivory and Marti A. Hearst. Improving web site design. *IEEE Internet Computing*, 6(2):56–63, 2002.

[Ire99] Langner Irene. An introduction to internet mailinglist research. In *Deutschsprachiger Japanlogentag in trier 1999*, volume 2, pages pp.653–665, 1999.

[Ivo00] Melody Y. Ivory. Web TANGO: Towards Automated Comparison of Information-centric Web Site Designs. In *CHI '00: CHI '00 Extended Abstracts on Human Factors in Computing Systems*, pages 329–330, New York, NY, USA, 2000. ACM.

[JA04] Timo Jokela and Pekka Abrahamsson. Usability assessment of an extreme programming project: Close co-operation with the customer does not equal to good usability. In *5th International Conference, PROFES '04*, pages 393–407, 2004.

[JF05] Plinio Thomaz Aquino Junior and Lucia Vilela Leite Filgueiras. User modeling with personas. In *CLIHC '05: Proceedings of the 2005 Latin American conference on Human-computer interaction*, pages 277–282, New York, NY, USA, 2005. ACM.

[Joo] Joost. Free online TV. http://www.joost.com. Last visit: 01.11.2007.

[Jum] Jumpcut. Be good to your video. http://www.jumpcut.com. Last visit: 01.11.2007.

[KK05] Eeva Kangas and Timo Kinnunen. Applying user-centered design to mobile application development. *Commun. ACM*, 48(7):55–59, 2005.

[KKSP06] Osamu Kobayashi, Mitsuyoshi Kawabata, Makoto Sakai, and Eddy Parkinson. Analysis of the interaction between practices for introducing xp effectively. pages 544–550, 2006.

[KMS05] Hendrik Knoche, John D. McCarthy, and M. Angela Sasse. Can small be beautiful?: Assessing image resolution requirements for mobile TV. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 829–838, New York, NY, USA, 2005. ACM Press.

[KS04] Jesper Kjeldskov and Jan Stage. New techniques for usability evaluation of mobile systems. *International Journal of Human-Computer Studies*, 60(5-6):599–620, May 2004.

[KS10] Jeff Sutherland Ken Schwaber. Scrum. Technical report, 2010.

[KT79] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):263–292, 1979.

[LB03] Craig Larman and Victor R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, 2003.

[LC05] Amy Law and Raylene Charron. Effects of agile practices on social factors. In *HSSE '05: Proceedings of the 2005 workshop on human and social factors of software engineering*, volume 30, pages 1–5, New York, NY, USA, July 2005. ACM Press.

[Lec05] Martin Lechner. Curid - a software development method for data-

driven framework architectures. Master's thesis, Technical University of Graz, Austria., 2005.

[Lec08] Martin Lechner. Xp team psychology - an inside view. In *PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference,*, 09 2008. PPIG 2008, The 20th Annual Psychology of Programming Interest Group Conference, Lancaster University, UK. 10th - 12th September 2008.

[lin] LinesOfCodeWichtel. www.andreas-berl.de/linesofcodewichtel/en-/index.html.

[LL08] Martin Lechner and David M. Loder. Evaluierung der plattform "neurovation" - eine untersuchung von erfolgsfaktoren in communities für die "neurovation" plattform. Technical report, TU Graz, Knowledge Management Institute, 2008. 2008.

[LP02] Stacy Lynch and Limor Peer. Analyzing newspaper content: A how-to guide. Technical report, Readership Institute, 2002.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[LT01] J. Luca and P. Tarricone. Does emotional intelligence affect successful teamwork. In *Proceedings of 18thAnnual Conference of the Australasian Society for Computers in Learning in Tertiary Education atthe ASCILITE, University of Melbourne, Melbourne*. Citeseer, 2001.

[LW04] William Krebs Laurie Williams, Lucas Layman. Extreme programming evaluation framework for object-oriented languages (version 1.4). Technical report, North Carolina State University, Department of Computer Science, 2004.

[LWST08] Michael Leitner, Peter Wolkerstorfer, Reinhard Sefelin, and Manfred Tscheligi. Mobile multimedia: identifying user values using the means-end theory. In *Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, pages 167–175, Amsterdam, The Netherlands, 2008. ACM.

[Man01] Agile Manifesto. Manifesto for agile software development. http://agilemanifesto.org/, 2001. visited: 12.03.2010.

[MB03] John Mendonca and Jeff Brewer. Lean, light, adaptive, agile and appropriate software development: the case for a less methodical methodology. pages 42–52, 2003.

[McG03] Gary McGraw. From the ground up: The dimacs software security workshop. *IEEE Security & Privacy.*, 1(2):59–66., March/April 2003.

[McL07] Laurianne McLaughlin. Next-generation entertainment: Video goes mobile. *IEEE Pervasive Computing*, 06(1):7–10, 2007.

[McN] Marc McNeill. User centred design in agile application development.

http://www.thoughtworks.com/pdfs/agile_and_UCD_MM.pdf.

[ML10] Sandra Dittenberger Michael Leitner. M3 - usability report usability study i january 2008. Technical report, CURE - Center for Usability Research and Engineering, January 2010.

[MM05] Paul McInerney and Frank Maurer. UCD in agile projects: dream team or odd couple? *Interactions*, 12(6):19–23, 2005.

[MN90] Rolf Molich and Jakob Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33(3):338–348, 1990.

[MOH07] Kjetil Molokken-Ostvold and Nils Christian Haugen. Combining estimates with planning poker–an empirical study. In *ASWEC '07: Proceedings of the 2007 Australian Software Engineering Conference*, pages 349–358, Washington, DC, USA, 2007. IEEE Computer Society.

[MRH07] Thomas Memmel, Harald Reiterer, and Andreas Holzinger. Agile methods and visual specification in software development: A chance to ensure universal access. In Constantine Stephanidis, editor, *HCII Universal Access in HCI*, volume 4554 of *LNCS*, pages 453–462. Springer, 2007.

[MS09] Lynn Miller and Desir\&\#233;e Sy. Agile user experience SIG. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems CHI*, pages 2751–2754, Boston, MA, USA, 2009. ACM.

[Nel02] Elden Nelson. Extreme programming vs. interaction design, 2002. FTP Online.

[NIA07] James E. Nieters, Subbarao Ivaturi, and Iftikhar Ahmed. Making personas memorable. In *CHI '07: CHI '07 extended abstracts on Human factors in computing systems*, pages 1817–1824, New York, NY, USA, 2007. ACM.

[Nie93] Jacob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1993.

[Nie00] Jacob Nielson. Why you only need to test with 5 users. http://www.useit.com/alertbox/20000319.html, March 2000. visited 13.6.2010.

[NL93] Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 206–213, New York, NY, USA, 1993. ACM.

[NNH06] Esa Nettamo, Mikko Nirhamo, and Jonna Häkkilä. A cross-cultural study of mobile music: retrieval, management and consumption. In *OZCHI '06: Proceedings of the 18th Australia conference on Computer-Human Interaction*, pages 87–94, New York, NY, USA, 2006. ACM.

[NWW+03] Nachiappan Nagappan, Laurie A. Williams, Eric Wiebe, Carol

Miller, Suzanne Balik, Miriam Ferzli, and Julie Petlick. Pair learning: With an eye toward future success. In *XP/Agile Universe*, pages 185–198, 2003.

[OF08] Hartmut Obendorf and Matthias Finck. Scenario-based usability engineering techniques in agile development processes. In *CHI '08*, pages 2159–2166. ACM, 2008.

[OMV07] Kenton O'Hara, April S. Mitchell, and Alex Vorbau. Consuming video on mobile devices. In *CHI '07: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 857–866, New York, NY, USA, 2007. ACM.

[PA06] John Pruitt and Tamara Adlin. *The Persona Lifecycle: Keeping People in Mind Throughout Product Design (The Morgan Kaufmann Series in Interactive Technologies)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[Pat02] Jeff Patton. Hitting the target: adding interaction design to agile software development. In *OOPSLA 2002 Practitioners Reports*, Seattle, Washington, 2002. ACM.

[PL08] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135, 2008.

[PMI04] Project Management Institute PMI. *A guide to the project management body of knowledge (PMBOK® guide)*. Project Management Institute, Newtown Square, PA, 2004.

[PP02] L. Paganelli and F. Paterno. Automatic reconstruction of the underlying interaction design of web applications. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 439–445, New York, NY, USA, 2002. ACM.

[PPS06] Fabio Paternò, Angela Piruzza, and Carmen Santoro. Remote web usability evaluation exploiting multimodal information on user behavior. In Gaëlle Calvary, Costin Pribeanu, Giuseppe Santucci, and Jean Vanderdonckt, editors, *CADUI*, pages 287–298. Springer, 2006.

[PT07] Marc Pifarré and Oscar Tomico. Bipolar laddering (bla): a participatory subjective exploration method on user experience. In *DUX '07: Proceedings of the 2007 conference on Designbiolaring for User eXperiences*, pages 2–13, New York, NY, USA, 2007. ACM.

[Rai07] J. B. Rainsberger. My greatest misses: Xp 2000-2007, agile 2007. August 18, 2007.

[Rhe00] Howard Rheingold. *The Virtual Community: Homesteading on the Electronic Frontier, revised edition.* The MIT Press, November 2000.

[RO01] Thomas J. Reynolds and Jerry C. Olson, editors. *Understanding Consumer Decision Making: The Means-end Approach to Marketing and Advertising Strategy.* Lawrence Erlbaum, Mahwah, NJ, 2001.

[RS02] Bernhard Rumpe and Astrid Schröder. Quantitative survey on extreme programming projects. In *In: Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, XP2002*, pages 26–30, 2002.

[Rub94] Jeffrey Rubin. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[SAHG06] Sharifah Syed-Abdullah, Mike Holcombe, and Marian Gheorge. The impact of an agile methodology on the well being of development teams. *Empirical Software Engineering*, 11(1):143–167, March 2006.

[SBLC+03] Jeffrey Sanchez-Burks, Fiona Lee, Choi, Incheol, Nisbett, Richard, Zhao, Shuming, and Jasook Koo. Conversing across cultures: East-west communication styles in work and nonwork contexts. *Journal of Personality and Social Psychology*, 85(2):363–372, 2003.

[Sch95] K. Schwaber. Scrum development process. In *OOPSLA Business Object Design and Implementation Workshop*, volume 27. Citeseer, 1995.

[Sch08] Christian Schindler. Agile software development methods and practices in austrian it-industry: Results of an empirical study. *Computational Intelligence for Modelling, Control and Automation, International Conference on*, 0:321–326, 2008.

[Sch10] Christian Schindler. *Review of Agile Software Development Methods in Practice*. PhD thesis, Graz University of Technology, Institute for Software Technology, 2010.

[Sma] Project Smart. The curious case of the chaos report 2009. http://www.projectsmart.co.uk/the-curious-case-of-the-chaos-report-2009.html. visited 10.03.2010.

[SMPJ06] Jan-Willem Strijbos, Rob L. Martens, Frans J. Prins, and Wim M. G. Jochems. Content analysis: what are they talking about? *Comput. Educ.*, 46(1):29–48, 2006.

[Sof] Libre Soft. Debian counting. http://libresoft.dat.escet.urjc.es/debian-counting/. last visited 10.02.2010.

[SPMW02] Giancarlo Succi, Witold Pedrycz, Michele Marchesi, and Laurie Williams. Preliminary analysis of the effects of pair programming on job satisfaction. In *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2002)*, Alghero, Sardinia, Italy, may 2002.

[ST05] Lori Shyba and James Tam. Developing character personas and scenarios: vital steps in theatrical performance and hci goal-directed design. In *C&C '05: Proceedings of the 5th conference on Creativity & cognition*, pages 187–194, New York, NY, USA, 2005. ACM.

[Ste01] Steve Stemler. An overview of content analysis. practical assessment, research & evaluation an overview of content analysis. Technical report, 2001.

[STG03] Reinhard Sefelin, Manfred Tscheligi, and Verena Giller. Paper prototyping - what is it good for?: a comparison of paper- and computer-based low-fidelity prototyping. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 778–779, New York, NY, USA, 2003. ACM Press.

[Stu05] E.C. Stubbs. *Emotional intelligence competencies in the team and team leader: A multi-level examination of the impact of emotional intelligence on group performance.* PhD thesis, Case Western Reserve University, Cleveland, 2005.

[Sut04] Jeff Sutherland. Agile development: Lessons learned from the first scrum. October 2004.

[svn] Svnstat. http://svnstat.sourceforge.net/. visited 17.06.2010.

[SY06] S.R. Subramanya and Byung K. Yi. User interfaces for mobile content. *IEEE Computer*, 39(4):85–87, April 2006.

[Sy07] D. Sy. Adapting usability investigations for agile user-centered design. *Journal of Usability Studies*, 2(3):112–132, 2007.

[Tes03] Bjørnar Tessem. Experiences in learning xp practices: A qualitative study, xp2003. In *XP*, pages 131–137, 2003.

[The95] Anuradha Gokhale The. Collaborative learning enhances critical thinking. *Journal of Technology Education*, 7, 1995.

[Tox05] Anders Toxboe. Introducing user-centered design to extreme programming. May 2005. http://blog.anderstoxboe.com/uploads/-16082005_XP_and_UCD.pdf.

[TVE] TVEyes. http://www.tveyes.com. Last visit: 01.11.2007.

[Usa] Usability.gov. Step-by-Step Usability Guide. http://www.usability.gov/. Last visited: 18.08.2008.

[W3C04] W3C. Notes on user centred design process (UCD). http://www.w3.org/WAI/EO/2003/ucd, April 2004. Last visited: 19.01.2009.

[web] Webster's revised unabridged dictionary (1913). http://www.encyclo.co.uk/webster/M/56.

[Web90] Robert P. Weber. *Basic Content Analysis (Quantitative Applications in the Social Sciences).* Sage Publications, Inc, 2nd revised edition edition, September 1990.

[Wes02] David West. Metaphor, architecture, and xp 101-104. In *Proceedings of the Third International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 101–104, May 2002.

[Whe02] David A. Wheeler. More than a gigabuck: Estimating gnu/linux's size (version 1.07). http://www.dwheeler.com/sloc/redhat71-v1/redhat71sloc.html, 2002.

[WHS+02] Sarah J. Waterson, Jason I. Hong, Tim Sohn, James A. Landay, Jeffrey Heer, and Tara Matthews. What Did They Do? Understanding

Clickstreams with the WebQuilt Visualization System. In *AVI '02: Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 94–102, New York, NY, USA, 2002. ACM.

[Wik05] Wikipedia: Learning theory (education) - visited 2005. http://en.wikipedia.org/wiki/Learning_theory_(education), 4 2005.

[Wil00] Laurie Ann Williams. *THE COLLABORATIVE SOFTWARE PROCESS*. PhD thesis, University of Utah, 2000.

[WK00] Laurie Williams and Bob Kessler. The effects of "pair-pressure" and "pair-learning" on software engineering education. In *CSEET '00: Proceedings of the Thirteenth Conference on Software Engineering Education & Training*, page 59, Washington, DC, USA, 2000. IEEE Computer Society.

[WK02] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Wol09] Peter Wolkerstorfer. Methodenvergleich: Mobile umfrage vs. tagebuch am beispiel des mamtam field trial. Technical report, CURE - Center for Usability Research and Engineering, 2009.

[WRLP94] Cathleen Wharton, John Rieman, Clayton Lewis, and Peter Polson. The cognitive walkthrough method: a practitioner's guide. pages 105–140, 1994.

[WSA04] Laurie Williams, Anuja Shukla, and Annie I. Anton. An initial exploration of the relationship between pair programming and brooks' law. pages 11–20, 2004.

[WTS+08] Peter Wolkerstorfer, Manfred Tscheligi, Reinhard Sefelin, Harald Milchrahm, Zahid Hussain, Martin Lechner, and Sara Shahzad. Probing an agile usability process. In *CHI '08: CHI '08 extended abstracts on human factors in computing systems*, pages 2151–2158, New York, NY, USA, 04 2008. ACM. CHI '08 Florence / Italy, April 5-10 2008 (http://www.chi2008.org/).

[WvD07] Ruud Wijnands and Ingmar van Dijk. Multi-tasking agile projects: The pressure tank. In *Agile Processes in Software Engineering and Extreme Programming*, volume 4536/2007 of *Lecture Notes in Computer Science*, pages 231–234. Springer Berlin / Heidelberg, 2007.

[wwwa] E-NOVATION gmbh. http://www.e-novation.eu/. visited 06.03.2010.

[wwwb] Grizzly werbeagentur graz. http://www.grizzly.cc. visited: 06.03.2010.

[wwwc] Kapsch carriercom ag. visited 06.03.2010. www.kapschcarrier.com.

[wwwd] Krone.tv. http://www.krone.tv. visited 06.03.2010.

[wwwe] Mobilkom austria. http://www.mobilkom.at/. visited 06.03.2010.

[wwwf] multimedia & e-business STAATSPREIS. http://www.multimedia-

staatspreis.at. visited 06.03.2010.

[wwwg] Orf. http://www.orf.at. visited 06.03.2010.

[wwwh] Sail labs technology ag. http://www.sail-technology.com. visited 06.03.2010.

[wwwi] W3C markup validaton service. visited 07.03.2010. http://validator.w3.org/.

[wwwj] www.softwaretop100.org. Top 10 fastest growing software companies. http://www.softwaretop100.org/software-top-100/fast-growth-top-10. visited: 12.03.2010.

[www10a] Cure - center for usability research and engineering. http://www.cure.at/, 2010. visited 07.03.2010.

[www10b] Softnet austria competence network. http://www.soft-net.at/, 2010. visited: 06.03.2010.

[Yaha] Yahoo group message archiver (ygma). http://sourceforge.net/projects/ygma/.

[Yahb] Yahoo. Extreme programming group. http://tech.groups.yahoo.com/group/extremeprogramming.

[Yip] Jason Yip. It's not just standing up: Patterns of daily stand-up meetings. http://martinfowler.com/articles/itsNotJustStandingUp.html.

[You] Mobile YouTube. http://m.youtube.com. Last visit: 01.11.2007.