

Technological support for distributed agile development

Kevin Dullemond and Ben van Gasteren

Technological support for distributed agile development

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Kevin Dullemond
born in Rotterdam



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

Technological support for distributed agile development

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Ben van Gameren
born in Rotterdam



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

Technological support for distributed agile development

Author: Kevin Dullemond
Student id: 1217445
Email: k.dullemond@student.tudelft.nl

Author: Ben van Gameraen
Student id: 1221752
Email: b.j.a.vangameraen@student.tudelft.nl

Abstract

Because of the distance between the dispersed development locations, Global Software development (GSD) is confronted with challenges regarding communication, coordination and control of the development work. At the same time, agile software development is strongly built upon communication between engineers and has proven its benefits, although, mostly on one single site. As such, it might be advantageous to combine GSD with agile development. This blend however is not straightforward since the distributed and agile development approaches might have conflicting convictions. In this thesis we will discuss the advantages and challenges of combining GSD with agile development based on a literature-based research. The main results presented in the theoretical part of this thesis (Part I through V), are: (i) aspects of agile software development, (ii) benefits and challenges associated with these in relation to GSD, (iii) categories of technological support for agile GSD, (iv) a framework depicting the mutual relations among them and (v) a discussion regarding specific technologies that support collaborative development in relation to this framework. Based on one of the recommendations we make in the theoretical part of this thesis we also perform practical research (Part VI) in which we define a list of requirements for an Integrated Collaborative Development Environment (ICDE) and show the technical feasibility of a number of concepts which realize these.

Thesis Committee:

Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: Dr. ir. D.M. van Solingen, Faculty EEMCS, TU Delft
Committee Member: Ir. B.R. Sodoyer, Faculty EEMCS, TU Delft

Preface

Research Division and Responsibilities

The research reported in this thesis is a joint effort of both authors of this thesis. To make the division of the research and responsibilities explicitly clear, this thesis is divided in parts. Here we will specify for each part by which author, or authors, it was written.

The actual research division and responsibilities are as follows:

Part I	Introduction	Kevin Dullemond & Ben van Gameren
Part II	Global Software Development	Ben van Gameren
Part III	Extending Global Software Development with aspects of the agile development process	Kevin Dullemond
Part IV	Supporting Agile Global Software Development with technology	Kevin Dullemond & Ben van Gameren
Part V	Conclusions and Future Work	Kevin Dullemond & Ben van Gameren
Part VI	Practical Research	Kevin Dullemond & Ben van Gameren

Division of the research assignment and the thesis project

The second year of the master Computer Science in the TU Delft consists of a research assignment and a thesis project. Often the research assignment consists of a literature study in preparation of the thesis project while the thesis project consists of solving a research or engineering problem. In this thesis the research assignment was indeed a literature study in preparation of the research project. Basically, the thesis project continued right where the

research assignment finished. Because of this we decided to present both in a single thesis.

In order for the supervisors to grade this work however, a split between both lines of research should be made. Because this thesis is structured as a single narration the split between both assignments is not explicit. On the one hand *Part I* and *Part II* are completely part of the research assignment, while *Part IV*, *Part V* and *Part VI* are completely part of the thesis project. *Part III* on the other hand cannot be completely be assigned to either of the two. In this part, the introduction into agile methodologies and the benefits and challenges of agile practices in a GSD environment are part of the research assignment. The definition of aspects of agile software development and how these aspects are related to distance and the challenges and benefits of agile practices in a GSD environment, however, are part of the thesis project.

Acknowledgements

We would like to express our gratitude to everyone who contributed directly or indirectly to our work. In this section we will mention the people who made the most profound contribution. First of all, we would like to thank our supervisor Rini van Solingen for his continuous support, guidance, enthusiasm, stimulating discussions, critical notes and review of our work. We would also like to thank him for his creative input with respect to deciding on the exact focus of our research. Secondly we would like to thank Arie van Deursen for the review of our work and his assistance with the L^AT_EX style. Thirdly during our research we have set up a small '*knowledge group*' with the help of Rini van Solingen.

The following people participate in this group:

Name	Company
Toine Hurkmans	Exact Software
Bart Platzbeecker	Exact Software
Dick Stegeman	iHomer
Floris van der Plas	Mavim
Ericka Marquez	SDL Tridion
Dennis van der Veeke	SDL Tridion
Onno Ceelen	SDL Tridion
Rini van Solingen	TU Delft
Gerard Janssen	Xebia

We would like to thank the participants of this group for the input and valuable discussions. We would also like to thank Rini van Solingen for approaching these people to cooperate in this format. Finally we would like to thank Kim Stehouwer for his help in the design of a number of figures and tables in this thesis.

Kevin Dullemond and Ben van Gameren
Delft, the Netherlands
June 7, 2009

Contents

Preface	iii
Research Division and Responsibilities	iii
Division of the research assignment and the thesis project	iii
Acknowledgements	iv
Contents	v
List of Figures	ix
List of Tables	xi
I Introduction	1
1 Introduction	3
1.1 Background and research focus	3
1.2 Scientific and societal motivation	4
1.3 Project structure	4
1.4 Thesis structure	6
2 Global Software Development	9
2.1 The general concept	9
2.2 Benefits	9
2.3 Challenges	11
II Global Software Development	15
3 The benefits of Global Software Development	17
3.1 Handle the increased product complexity	17
3.2 Usage of specialized or skilled people	18

3.3	Access to a sufficiently large workforce	18
3.4	Increased merger and acquisition possibilities	18
3.5	Global presence	18
3.6	Cost reduction of development	19
3.7	Reduction in time to market	19
3.8	Proximity to the market	19
3.9	Handle the increased organization scale	20
3.10	Overview	20
4	The challenges of Global Software Development	23
4.1	Geographic dispersion	23
4.2	Control and coordination breakdown	24
4.3	Loss of communication richness	25
4.4	Loss of teamness	26
4.5	Cultural differences	29
4.6	Overview	33
5	Non technological support for alleviating distance in global software development	35
5.1	Reduce intensive collaboration	35
5.2	Reduce cultural distance	36
5.3	Reduce temporal distance	38
III	Extending Global Software Development with aspects of the agile development process	39
6	Agile methodologies	41
6.1	What are agile methodologies?	41
6.2	Scrum	45
6.3	eXtreme Programming	48
6.4	Other agile methods	53
6.5	Aspects of agile software development	56
7	Global software development combined with aspects from agile methodologies	67
7.1	How aspects of agile methods can reduce distance in a GSD context	68
7.2	Problems with incorporating agile aspects into GSD	71
7.3	Overview	76
IV	Supporting Agile Global Software Development with technology	81
8	Types of technology which support GSD	83
8.1	Types of technology which support GSD in general	83
8.2	Types of technology which support agile GSD	88

9	How the different types of technological support are applicable to support agile GSD	91
9.1	An approach to derive technological support for agile GSD	91
9.2	How each aspect can be supported by the different types of technology . . .	92
9.3	Overview	100
10	How to support the incorporation of agile aspects into the GSD process with technology	105
10.1	Communication versus Software Development related technology	105
10.2	Collaborative technologies	107
10.3	Integration of collaborative technologies	114
V	Conclusions and Future Work	117
11	Conclusions and Further research	119
11.1	Contributions	119
11.2	Conclusions	120
11.3	Reflection	121
11.4	Recommendations for further research	122
VI	Practical Research	125
12	Practical Work	127
12.1	Research design	127
12.2	Context	129
12.3	Findings	133
12.4	Feasibility study	140
12.5	Validity of the practical work	184
12.6	Conclusions and recommendations for further research	184
	Bibliography	187
A	Interview Structure	199
B	Requirements of an ICDE	203

List of Figures

1.1	The structure of this thesis	7
5.1	Alternative paths to alleviating intensive collaboration [30]	36
5.2	A taxonomy of structural arrangements for software development [30]	37
6.1	Difference in Agile and Heavyweight Methodologies [8]	44
6.2	Difference in Agile and Heavyweight Methodologies [23]	45
6.3	The general phases of Scrum [132]	47
6.4	The Scrum sprint cycle [36]	47
6.5	The life cycle of the XP process [1]	52
6.6	The ASD cycle [79]	54
6.7	The five phases of FDD [37]	55
9.1	Relationships between aspect A_1 and the distances faced in GSD	92
9.2	Relationships between aspect A_2 and the distances faced in GSD	94
9.3	Relationships between aspect A_3 and the distances faced in GSD	96
9.4	Relationships between aspect A_4 and the distances faced in GSD	98
9.5	Relationships between aspect A_5 and the distances faced in GSD	99
9.6	Complete overview of relationships aspects and distances	101
9.7	Overview of the relationships between aspects and distances that can be supported by technological support	102
10.1	Communication related technologies	110
12.1	Presence dimensions	142
12.2	Presence in Office Communicator	142
12.3	Presence information of one of your contacts	145
12.4	Available communication options in Office Communicator	145
12.5	The conversation window in Office Communicator	146
12.6	Team Foundation Server 3-tier architecture	148
12.7	Sprint Burndown Chart	149

LIST OF FIGURES

12.8	Product Burndown Chart	150
12.9	OCS class diagram	152
12.10	Relational Database Schema of the OCS Data collector	154
12.11	Entity Relationship Diagram of the OCS Data collector	154
12.12	Relational Database Schema of the OC Actuator	156
12.13	Entity Relationship Diagram of the OC Actuator	156
12.14	TFS Data Collector Component class diagram	157
12.15	Relational Database Schema of the TFS Work Items	159
12.16	Entity Relationship Diagram of the TFS Work Items	160
12.17	Relational Database Schema of the TFS Version Control	161
12.18	Entity Relationship Diagram of the TFS Version Control	162
12.19	Relational Database Schema of the TFS Reports	163
12.20	Entity Relationship Diagram of the TFS Reports	163
12.21	Relational Database Schema of the TFS Data Collector Component	164
12.22	Entity Relationship Diagram of the TFS Data Collector Component	165
12.23	The design of the User Interface	167
12.24	Example of a Thickbox	167
12.25	Relational Database Schema of the Total System	168
12.26	Entity Relationship Diagram of the Total System	169
12.27	The menu-bar of the User Interface	170
12.28	The users list in the User Interface	170
12.29	The conversations list in the User Interface	171
12.30	The Sprint Backlog	172
12.31	The Product Backlog	173
12.32	The Reports	173
12.33	The User Data Item Page	174
12.34	The History Data Item Page	176
12.35	The Project Data Item Page	177
12.36	The Sprint Data Item Page	178
12.37	The Sprint Backlog Item Data Item Page	179
12.38	The Product Backlog Item Data Item Page	179
12.39	The Sprint Burndown Chart	180
12.40	The Product Burndown Chart	181
12.41	The join screen	182
12.42	The incoming join request pop-up	182
12.43	The popup screen to select a new leader	182
12.44	The pop-up screen notification that you are the new leader of a conversation	183

List of Tables

3.1	The benefits of GSD and their level of influence	21
4.1	The challenges of GSD and their classification	34
6.1	Relation between the Agile Manifesto and the aspects of agile software development	64
6.2	Relation between eXtreme programming and the aspects of agile software development	65
7.1	Benefits of incorporating agile aspects into GSD work	77
7.2	Challenges faced when incorporating agile aspects into GSD work	78
7.3	Proposed solutions to the challenges faced when incorporating agile aspects into GSD work	79
8.1	Technological opportunities to exploit the benefits of GSD	85
8.2	Technological opportunities to alleviate the challenges of GSD	86
8.3	Technological objectives to support GSD	87
9.1	Overview of technological support for aspect A_1	93
9.2	Overview of technological support for aspect A_2	96
9.3	Overview of technological support for aspect A_3	97
9.4	Overview of technological support for aspect A_4	98
9.5	Overview of technological support for aspect A_5	99
9.6	Overview of technological support for all benefits and challenges	103
9.7	Overview of technological support for all aspects	104
10.1	Relatedness of requirement categories	106
12.1	Mapping of development activities with their associated Knowledge Areas . .	135
12.2	The different states of the presence button	144

Part I

Introduction

Chapter 1

Introduction

The aim of the introductory chapter is to present our problem analysis and to establish objectives for this thesis by outlining our frame of research. First some background information is given and the research focus is defined. Then the research is justified, both scientifically and societally. Finally the overall structure of the project is provided. Here both the research assignments and the project assignment are introduced.

1.1 Background and research focus

Global software development (GSD), which is also known as global software engineering (GSE), globally distributed software engineering (GDSE) and globally distributed software development (GDSD) is becoming increasingly interesting these days due to the globalization of business [29, 74, 19, 60, 76, 46, 70, 120, 3]. In GSD the software development process is distributed between several geographically dispersed locations [43, 46, 130]. Advantages of GSD include: market-proximity [64, 73, 46], reducing time-to-market by working around the clock [29, 71, 50, 46], flexibility with respect to business opportunities [29, 71], reducing costs by delegating work to countries with low labor cost [30, 46] and being able to fully utilize available resources [74, 64, 46]. Besides being beneficial, GSD introduces a number of challenges in relation to communication, coordination and control of the development process. Examples are: lack of informal communication [29, 71, 74, 4], reduced hours of collaboration [11, 90, 83, 3], communication delay [4, 75, 70, 43] and loss of cohesion [29, 72, 70].

The cause of the challenges originating from GSD is the introduction of distance between the different actors involved in the software development process [82]. It is expected that this situation can be improved upon by combining agile software development with GSD [82, 109]. The agile software development approach is used often in practice [129] and is defined by Abrahamsson et al. [1] as an incremental, cooperative, straightforward and adaptive approach. Advantages of this approach are being able to respond to change [23], producing rapid value [23] and facilitating for closer collaboration, both in the development team and with respect to the customer [40].

The advantages of both these development approaches seem to complement each other, so blending them could result in a flexible and dynamic development approach which works in a distributed setting. Paasivaara et al. [109] however suggests that: *"The combination of agile methods and distributed development poses several challenges"*. This is likely since the distributed and agile development approaches seem to have conflicting convictions with regard to certain aspects of development. Agile development, for example, focuses mainly on informal processes and distributed development usually focuses on formal mechanisms [125]. These conflicts result in several challenges which should be dealt with for the combined development approach to be a feasible approach. In this thesis we will research the advantages and challenges of combining agile software development and GSD and how best to support these. In this research our focus will lie on achieving these goals by means of technological support. The main research question this graduate project will attempt to answer is: *"What are the advantages and challenges of the combination of the agile and distributed development approaches and how is technological support best used to deal with these?"*

1.2 Scientific and societal motivation

As mentioned in the previous section, both GSD and the agile development approach are used often in practice and possess certain qualities. The combination of both approaches is desirable to both the scientific world and society. This is because the market demands that software is delivered more quickly, while at the same time, many software development organizations have become dispersed, with more than fifty percent of their developers spread across multiple teams at geographically distributed sites [6]. There is also a need, in GSD, for the development to be flexible and to be able to accommodate the differences among the development teams [130]. These are qualities an agile development approach offers [1].

Research regarding the combination of GSD and agile software development is also requested in existing literature. For one Ramesh et al. [125] mentions it: *"careful incorporation of agility in distributed software development environments is essential in addressing several challenges to communication, control, and trust across distributed teams"*. Secondly one of the largest pitfalls in distributed development are issues related to communication [130] and Paasivaara et al. [109] suggests these issues could be dealt with by applying practices from agile development. Paasivaara et al. [108] also suggests researching tool support for projects of this kind is interesting. Finally, the literature on using agile methods in global software development is still scarce [109]. All these reasons lead to the conclusion that further research is justified.

1.3 Project structure

The project will start with two literature studies. One will investigate the available supporting technologies for distributed development while the other will focus on eliciting the

challenges and benefits of combining the agile and distributed development approaches. Ben van Gasteren will conduct the former, while Kevin Dullemont will conduct the latter of these two literature studies. These studies will then be combined to determine how best to deal with these challenges and how to exploit the benefits as much as possible. In this section we will introduce both research assignments and conclude with explaining why we elected to conduct this project as a joint effort.

1.3.1 Research assignment 1

Technologies which support GSD are needed by globally positioned organizations. In this context the term technology can refer to both a methodology and a tool. This research assignment will consider the different technologies which are commonly accepted, it includes the subtopics communication- and involvement- facilitating technologies and addresses the limitations of these technologies. The main research question of this assignment will be: *"Which technologies are available to support distributed development and what possibilities and limitations do they possess?"*

1.3.2 Research assignment 2

Combining the agile and distributed development approaches will result in certain benefits and challenges. For the combined development approach to be optimal the challenges should be faced and the benefits should be fully exploited. In order to attempt this, first the challenges and benefits must be determined. So the main research question of this research assignment will be: *"To which extent is it desirable to combine the agile and distributed development approaches?"* The three questions that need to be answered to answer the main research question are:

1. *"What are the benefits of combining the agile and distributed development approaches?"*
2. *"What challenges arise when combining the agile and distributed development approaches?"*
3. *"How can the challenges, that arise when combining the agile and distributed development approaches, be faced?"*

1.3.3 Justification of the joint research effort

This project was conducted by two master students as a joint effort because this poses several benefits. For one, because the two research questions are related, the amount of duplicate research performed is reduced, leaving more room to put research time towards innovative ideas and accomplish more than either of the researchers could have accomplished on their own, in the time available. Next to this, because both research problems collectively support a single objective; determining how best to support agile GSD with technology, joining them together results in more than the individual parts. The idea is that this combination is sufficient to reach a founded understanding of the opportunities of agile

GSD and how best to support these. Finally, the conclusions of this research will benefit from the complementary views of both researchers regarding the subject.

1.4 Thesis structure

The overall structure of this thesis is given in Figure 1.1. In Part I what is to be researched in this project is defined and the concept of global software development is introduced. In part II global software development is further explored leading to a list of challenges and a list of benefits associated with GSD in general. In this part also the non-technological support which can be used to respectively alleviate and exploit the aforementioned challenges and benefits. In part III the second line of research explores the concept of agile software development and links this to GSD. This is done by defining aspects of agile software development and then discussing both how these aspects help and, at the same time, are harder to realize when working globally distributed. This part also discusses procedural ways to deal with the difficulties with incorporating the aspects of agile software development in the GSD process. In part IV the two lines of research come together to discuss technical approaches to support agile GSD. First five categories of technological support which are useful to agile GSD are defined. This is followed by discussing for each aspects of agile GSD how the corresponding benefits and challenges can best be exploited or dealt with, with the help of technology. Subsequently, in part V, the research is concluded by discussing the answers to the research questions and by making recommendations with respect to further research on this subject. Finally, in part VI, a practical research is performed bases on a recommendation made in part V. In this part a list of requirement for an Integrated Collaborative Development Environment (ICDE) is defined and the technical feasibility of a number of concepts which realize these, is shown.

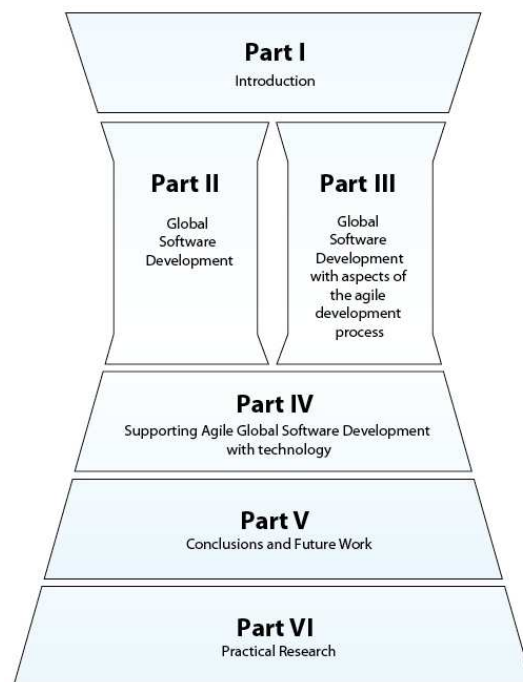


Figure 1.1: The structure of this thesis

Chapter 2

Global Software Development

The aim of this chapter is to introduce the concept of global software development (GSD). To do this, the general concept of GSD is discussed, followed by a general discussion of both the advantages and challenges associated with this development approach.

2.1 The general concept

For the last couple of decades the way business is being conducted has been becoming more and more global and this increases the pressure to distribute software projects globally [29, 74, 19, 60, 76, 46, 70, 120]. This way of approaching software development is known by many names. One of the most popular is global software development (GSD), which will be used in this thesis. Other terms which are used in the literature are: global software engineering (GSE), globally distributed software engineering (GDSE) and globally distributed software development (GDSD). In this approach the development process is distributed between several geographic locations [76, 140, 43, 46, 130]. Besides the globalization trend, also the uprise of advanced information and communication technologies has enabled new options, allowing distributed work to become easier and more efficient [10, 49, 130, 3].

2.2 Benefits

In addition to the globalization trend and the uprise of advanced information and communication technology, there are other reasons for performing projects in a distributed setting [130]. In fact, major benefits have been attributed to GSD, in spite of problems it might cause [3]. The most well known sources that mention benefits are Carmel [29] and Herb-
sleb et al. [74]. Carmel distinguishes four categories of benefit factors, namely [29]:

1. *Catalyst factors*
Reasons for starting to use a distributed development approach.
2. *Sustainment factors*
Reasons for continuing to use a distributed development approach.

3. *Size factors*

Reasons for using a distributed development approach originating from size implications of the organization.

4. *Vision factors*

Reasons that have to do with a certain vision of future events. Because of choosing to follow the distributed development approach now, certain positive future developments could emerge. Benefit factors from this category can be seen as catalyzers for innovation.

Jacobs et al. [84] made a minor extension by incorporating benefits from Lipnack [97] and Moll et al. [139]. Next, the most important benefits of GSD found in the literature will be briefly introduced.

- *Product complexity*

Software systems become more and more complex due to the continuously growing possibilities provided by technology and the wider application of these possibilities [139, 89]. Because of this; single projects, single departments or even single companies can no longer develop total products. By concurrent and distributed development the product complexity is handled at the cost of added organizational complexity [139].

- *Usage of specialized or skilled people*

Highly skilled engineers, software professionals and managers are a scarce commodity [7] so companies want to make optimal use of them, regardless of their geographical location [29, 74].

- *Access to a sufficiently large workforce*

There is a general shortage of people capable of developing software of sufficient quality [9, 51, 101, 130]. By offshoring companies get access to a labor pool which is much larger [29, 30, 51, 31, 130].

- *Acquisitions*

By being able to cooperate between geographically dispersed locations, a company becomes more flexible with respect to merger and acquisition opportunities wherever they present themselves. [29, 74]

- *Global presence*

By having development centers all over the world, the corporate image of companies includes being a global player. [29]

- *Cost-reduction of development*

By shifting work to countries where the cost of labor is lower, the total development cost can be reduced [29, 30, 50, 74, 75, 3].

- *Reduction in time-to-market*
By dividing development across multiple time zones follow-the-sun or round-the-clock development can take place, which can reduce the time to market of a product. [29, 30, 50, 74, 3]
- *Proximity to the market*
By developing software physically close to the customer certain business advantages emerge such as an increase of knowledge of the local market [29, 74] and the creation of good will by investing in the local market [51]. Another important benefit of being close to the customer is the ability to respond to local circumstances and preferences [134]
- *Organization scale*
Above a certain size, development centers get too difficult to manage. By dividing the work across several, smaller and thus more flexible, units the management problem of the single development center is taken care of [29].

Having introduced the most important benefits of GSD we feel it should be noted that the benefits are not universally applicable. Both Conchúir et al.: *"While there are many significant beneficial aspects of GSD, our study clearly shows that these benefits are neither clear-cut nor can their realization be as taken-for granted as the GSD literature may lead one to believe"* [44] and Gumm: *"What is perceived as benefit in one case, may be only partially realizable, a disadvantage or even a myth in another one"* [66] state this. The general applicability of these 'potential' benefits will be discussed further in chapter 3.

2.3 Challenges

Besides being beneficial, GSD also introduces a number of challenges in relation to communication, coordination and control of the development process. These challenges arise due to geographical, temporal, and socio-cultural distances associated with developing in a distributed setting [106, 43]. These distances are defined by Carmel as the three unique aspects of GSD [29]:

- *Geographical distance* between development sites has a direct impact on project control, coordination, and communication.
- *Temporal distance* between development sites make it harder to communicate, impacting project control and coordination.
- *Cultural distance* between employees of the development sites may lead to mistrust, miscommunication and lack of cohesion.

Based on these distances Carmel identifies the following five problem areas that potentially threaten the delivery of the product in time within budget, and with the specified or implicitly expected product quality:

- *Geographic dispersion*

Geographical dispersion reduces the frequency of informal communication as opposed to face-to-face work [71, 73, 83]. In fact the frequency of communication drops very sharply with physical separation and when the distance has increased to thirty meters, further increasing the separation has very little influence [5, 73]. Holmström et al. [83] suggest that measuring geographical distance, in the context of GSD, is best measured in ease of relocation rather than kilometers since a small distance in kilometers can imply a very large communication difficulty and vice versa. In comparison with face-to-face work, when working geographically dispersed communication lines are longer, it is harder to give feedback quickly, creating and maintaining trust is more difficult and miscommunication is more frequent [124].

- *Control and coordination breakdown*

Carmel sees control as the process of adhering to goals, policies or standards and coordination as the act of integrating each task and organizational unit so that it contributes to the overall objective. The overhead of control and coordination associated with any software project is large and it increases with GSD because of the reduced amount of informal communication [29].

- *Loss of communication richness*

Besides reducing the frequency of communication, using the distributed development approach also reduces the richness of communication [73]. Carmel defines rich communication as two-way interaction involving more than one sensory channel [29]. Following this definition face-to-face is the richest communication medium [35, 29]. So the inability to communicate face-to-face and the resulting decline in communication richness and subtle interaction, results in considerable challenges [73, 106]. Loss of communication richness is considered a threat to communication, collaboration, and trust [84].

- *Loss of teamness*

In good teams, team members help and complement each other and know the other team members well. In such teams there is also high cohesion which leads to enhanced motivation, enhanced moral, greater productivity, harder work, more open communication, and higher job-satisfaction [29]. Because of the distances in geographical location and culture, as well as the loss of communication richness, the development teams possess less of these properties. Herbsleb et al. also show [73]: *"a significant relationship between delay in cross-site work and the degree to which remote colleagues are perceived to help out when workloads are heavy. This result is particularly troubling in light of the finding that workers generally believed they were as helpful to their remote colleagues as to their local colleagues."*

- *Cultural differences*

In GSD projects people from different areas of the world work together. When people from different cultures have to work together this can cause problems because their customs and beliefs can be conflicting. Examples of such problems are: it

takes more time to reach consensus, the chance of misunderstandings increases and building a cohesive team is more difficult [29].

The challenges from the problem areas discussed are still hard to deal with and will be discussed further in chapter 4. There are two basic approaches to target these challenges. The first is to use certain procedures to deal with them, this is discussed in chapter 5, and the other is to make use of technological support; this is discussed in chapter 8.

Part II

Global Software Development

Chapter 3

The benefits of Global Software Development

The most important benefits of GSD have already been roughly introduced in the previous chapter. In this chapter a more detailed description of these benefits is given and the general applicability will be discussed. In each of the next sections a benefit is presented and examined thoroughly.

3.1 Handle the increased product complexity

Products become more and more complex nowadays, both from technical and managerial standpoints, due to the continuously growing possibilities provided by technology and the wider application of these possibilities [139, 89]. This development is especially seen in industrial products and consumer electronics such as DVD recorders, electron microscopes and microwaves. These kinds of products are constructed from a large number of subsystems or components. The development of such complex products is imposed with three main constraints [139]:

1. Complex products are increasingly confronted with higher quality demands
2. Market pressure forces complex products to be developed faster
3. The development of complex products results in equally complex organizational issues

Due to these constraints, the development of these kinds of products can become so complex that a single department, or even a single company, is not capable to deliver these products anymore [139]. In order to overcome the organizational, technical and financial issues that arise from the development of complex products, the product development can be divided into separate and concurrent projects. These projects are responsible for a part of the product and can be performed in a geographically dispersed environment [50]. The partitioning of work tasks horizontally also implies that each site is responsible for a particular function or module. This subdivision makes it possible to develop a complex product in a distributed

setting but causes overhead for project organization, project management and engineering [99].

3.2 Usage of specialized or skilled people

The explosive growth of the software industry has led to a shortage of highly skilled engineers, software professionals and managers [7]. This situation makes it clear that organizations can no longer rely on local skilled labor only, they must have a broader view. In order to be competitive, software development companies must deploy the best software designers and developers in the world, regardless of their geographic location [29, 74, 3]. When a company is globally distributed it is easier to contract these skilled workers.

3.3 Access to a sufficiently large workforce

The demand for software engineers has grown substantially for the last decades, this makes it hard for companies to contract sufficient employees which are capable of developing software of sufficient quality at the existing site [9, 51, 50, 101, 130]. In the late 1990s the recruitment of new employees was very problematic due to the technology boom, the Y2K remediation efforts and the euro currency conversion [31]. Meanwhile, the number of graduates from universities and technical schools in India, China and other nations has increased [31]. As a consequence, organizations operate on a global platform to get access to a labor pool which is much larger [29, 30, 51, 50, 74, 31, 130, 3].

3.4 Increased merger and acquisition possibilities

Growth is one of the most important factors for a software company, because smaller firms cannot survive against software companies who possess marketing clout and diverse product offerings [29]. In order to increase their market share software development organizations make use of merger and acquisition opportunities [29, 74]. The last years mergers and acquisitions become a common practice, the global mergers for 2006 even reached an amount of USD 3.8 trillion [111]. Another advantage of mergers and acquisitions is the ability to reach new markets, develop new products and complement existing product lines [73, 50]. An international dispersed organization is more flexible than a single site organization with respect to merger and acquisition possibilities wherever they present themselves [29, 74].

3.5 Global presence

From a strategic point of view it can be beneficial for an organization to position them as a global player. Organizations which have their development sites all over the world are selling their products to two main groups, global businesses and global consumers. A global business prefers a large and established software supplier for all their global needs, rather than multiple smaller suppliers in different countries. Individual software consumers

follow the mass and prefer a product with a global image. By having software development centers all over the world a software company shows its worldwide aspirations [29].

3.6 Cost reduction of development

One of the most well known benefits of GSD is the potential to reduce the software development and maintenance costs [29, 30, 50, 74, 75, 43, 3]. The largest expense for a software company is the long-term maintenance of their applications in production. This includes the people, the infrastructure and the tools required to keep the software operational [111]. The difference in wages across different countries, between persons with equivalent skills, can be significant, with programmers in India earning roughly ten to twenty percent of what programmers in the United States earn [29, 3]. This difference in wages, is one of the reasons that companies consider globalizing their software development activities [3]. The globalization of software development and maintenance has been made possible by the deployment of cross-continental high-speed communication links, which make it possible to transfer the source code instantaneously [3]. The potential cost advantages can only be achieved by companies that are offshoring and outsourcing in a well-defined and planned way. If this is not the case; additional managerial overhead is required, perceived threat from lower paid colleagues can arise, and additional time to build up a '*critical mass*' is needed [43]. These cost-benefit tradeoffs are still not well understood and needs to be examined [52].

3.7 Reduction in time to market

"Given time zone differences, the ideal dispersed project can be productive around the clock" [29].

To reduce the time to market, organizations search for development approaches which could accelerate the development process. If a company has development centers in different countries, at different time zones all around the world, then it is possible to adapt the follow-the-sun approach. Follow-the-sun development, also known as round-the-clock development, has the potential to collapse time to market for project completion [29]. This becomes possible because dispersed software teams can work in 24-hour shifts, creating a '*virtual workday*'. At the end of the working day, one team passes the work to another team located in another time zone [29, 111]. Even in development project that do not operate during the complete 24 hours of the day, it is attractive to have developers working at one site while developers of another site sleep [52]. This approach can aid organizations which are under severe pressure to improve time-to-market [29, 30, 50, 74, 3].

3.8 Proximity to the market

By establishing subsidiaries near to the customer, software organizations meet the maxim to stay close to the customer [29]. Due to this dispersion a more direct interaction between

client and developer becomes possible [64, 73, 74]. This is particularly important when rich communication and relationships are needed between customers and developers, in the case of requirements gathering and design [29]. Another advantage of developing software physically close to the customer is the increased knowledge of the local market [29, 74]. By contracting local employees there do not exist cultural and linguistic distances to the customer, there is even a better knowledge of the local business conditions [29, 74, 43]. Creating new jobs can also create good will with local customers, possibly resulting in more contracts [29, 51]. When a company decides to contract local employees there will be a cultural division among team member, which introduce socio-cultural challenges. Finally, the ability to respond to local circumstances and preferences is a major benefit of being close to the customer [134].

3.9 Handle the increased organization scale

As stated in section 3.4, growth is one of the most important factors for a software company. As a consequence, software companies have grown quite large. This type of development centers now have well over thousand employees and become very hard to manage, it is even possible that at some point co-located development centers become too large and too difficult to manage [29]. By dividing the work across several, smaller and thus more flexible, units the management problem of the single development center is taken care of at the cost of coordination overhead across the dispersed sites [29].

3.10 Overview

In this section an overview of the benefits of global software development and their field of influence is presented. These benefits can have influence on the corporate level and on the project level. The project level embraces all the facets of a software development project such as informal communication, the sense of teamness, knowledge sharing etcetera. All the aspects which have influence on the decision making at an organizational level are embraced by the corporate level. Table 3.1 provides this overview in a structured manner; a double plus in the table indicates that the benefit at hand has a major impact on the corresponding level; a single plus denotes that the benefit has limited influence on the corresponding level; an empty cell denotes that the benefit has no influence at all on the corresponding level. For all the benefits, which are discussed in this section, a classification is sown. So, for example, the benefit of being able to contract specialized or skilled people all over the world has a major impact on the project level because development issues can be resolved much faster as a result of the increased amount of knowledge available in the project team. This benefit has limited impact on the corporate level, in contrast to the project level, because only some hiring procedures need to be changed in order to contract the appropriate people.

<i>Benefit</i>	<i>Corporate level</i>	<i>Project level</i>
Handle the increased product complexity	+	++
Usage of specialized or skilled people	+	++
Access to a sufficiently large workforce	+	++
Increased merger and acquisition possibilities	++	
Handle the increased organization Scale	++	
Global Presence	+	
Cost reduction of development	++	
Reduction in time to market	++	++
Proximity to market	++	+

Table 3.1: The benefits of GSD and their level of influence

Chapter 4

The challenges of Global Software Development

In contrast with the benefits presented in chapter 3, global software development also introduces a number of challenges in the field of communication, coordination and control of the development process. These challenges arise due to geographical, temporal and socio cultural distances associated with developing in a distributed setting [29, 106, 43, 83]. Based on these distances Carmel identifies five centrifugal forces [29]:

- Geographic dispersion
- Control and Coordination breakdown
- Loss of communication richness
- Loss of teamness
- Cultural differences

These five problem areas pull apart the global software team, having a negative influence on the development of the product. In the remaining of this chapter we describe and classify the challenges of global software development into their corresponding problem areas. In section 4.1, the challenges arising from geographic dispersion are presented. Secondly, in section 4.2 the control and coordination breakdown challenges are discussed, followed by the challenges originated from the loss of communication richness. The central issue in section 4.4 is the loss of teamness arising from the distributed development approach. In section 4.5 the cultural differences are covered. Finally, in section 4.6, an overview of all the challenges and their classification into one or multiple problem areas is presented.

4.1 Geographic dispersion

In this section the adverse affects of distance on the frequency of communication are discussed. In 1977 Allen showed a relationship between distance and communication, when

the distance between colleagues increases communication drops fast [5]. Other disadvantages of working geographically dispersed is that communication lines are longer, it is harder to give feedback quickly, creating and maintaining trust is more difficult and miscommunication is more frequent [124, 29].

Lack of informal communication

One of the main challenges which occur in a distributed setting is the lack of informal communication due to geographic or temporal separation [29, 71, 74, 4]. Especially in organizations with rapidly changing environments and dynamic projects informal communication plays a major role. Informal communication allows team members to develop working relationships, and allows a better flow of information about the current project [72]. The frequency of informal communication drops very sharply when colleagues are physical separated [71, 73, 83]. Allen found that this is the case when colleagues offices are more than 30 meters from another [5].

Increased effort to initiate contact

When developers are co-located, contact can generally be initiated quite easily. Employees know who is around and how busy they are. When team members are separated geographically or temporally, the effort required to initiate contact increases [71, 70, 3]. This is because developers are not aware of the skills and roles of their remote colleagues. One of the main difficulties to initiate contact is to determine the appropriate colleague [71, 70]. As a consequence it can be the case that a developer applies minor modifications to the system without trying to make contact with the person who has more knowledge of that part of the system; this can lead to errors in the system which slowdown the project.

4.2 Control and coordination breakdown

In this section the discussion of dispersion versus co-location highlights that it is harder to manage from a distance. Several communication, coordination and control mechanisms fall apart when team members are far apart. In such a setting it is not possible to walk to a colleague his office to discuss and resolve a problem immediately [29].

Reduced hours of collaboration

Having developers located in different time-zones allows organizations to increase the number of working hours during a day. An obvious disadvantage of being separated by temporal distance is that the number of overlapping working hours during a workday is reduced [11, 90, 83, 3]. In order to improve the effectiveness of the software development it is necessary to have some overlapping work-hours during the day [43]. In these overlapping hours it is possible to communicate directly with each other. However, sufficient overlap in working hours may be difficult to achieve due to, different working hours, lunch breaks and holidays [71, 43]. Only one hour time difference between two sites can already have a

large influence on the number of overlapping hours. There is an hour lost at the beginning and the end of each day, additionally there may be two hours lost since typical lunch time was displaced by an hour. So, a small time difference may already cause that there are only five overlapping hours in a day [71]. This means that team members might have to work flexible hours in order to increase the number of overlapping hours [96].

Lack of shared understanding

Software development requires much communication, both formal as informal, for handling crucial tasks. Informal communication helps people to stay aware of what other people are working on, what the current state of the project is, where the required expertise is located and other background information. All this information together enables developers to work together efficiently [74]. When an organization is globally dispersed this important project information must be shared differently. Managers must be able to adequately share import project information to all teams in order to exploit the benefits of GSD. However, when teams have inadequate information about the project they cannot determine what tasks are on the critical path and reuse opportunities may be overlooked [29, 74]. In order to exploit the benefits of GSD, there must be effective mechanisms for sharing information and facilitating common understanding.

Increased dependency on technology

An dispersed organization is dependent on information and communication technologies in order to collaborate. These technologies are used for communication and have impact on the coordination of the most critical processes in an organization [4]. Ebert et al. state that it is necessary to have a convenient and well working technical infrastructure for information and communication in order to run a global organization successful [50].

Increased complexity of the technical infrastructure

An unexpected challenge can arise when companies are using a variety of tools and products from third-party vendors. Obtaining global support for these tools can be a problem, just as obtaining the same version of the tools. Battin et al. found that software vendors are offering different versions of the tools in different countries. It can be possible that one site has the latest version of the tool while another site has an older version which are not completely compatible [11, 74, 131]. Finally the import and export regulations of each county must be understand totally, because these may prohibit the usage of certain technology throughout the distributed team [11].

4.3 Loss of communication richness

Using the distributed development approach not only reduces the frequency of communication as discussed before. But has also a negative impact on the richness of communication [73]. Rich communication is defined by Carmel as two-way interaction involving more

than one sensory channel. Following this definition face-to-face is the richest communication medium [35, 29]. The inability to communicate using rich communication media results in several challenges.

Communication delay

When organizations are dispersed across different time zones it is not always possible to contact a colleague at the moment that their help is needed. It is possible that this person is not at work or in an important meeting. In this case it is not possible to communicate directly; this implies that the use of asynchronous tools is required to be able to communicate. These kind of tools have the disadvantage that the amount of time it takes to receive a response increases [4, 75, 70, 43]. The delay in receiving a response from the remote site can increase the amount of time it takes to resolve the issue at hand [24]. Another reason for the delayed feedback is the increased risk of misunderstandings especially when the content of the message was ambiguous [47, 90].

4.4 Loss of teamness

Before the loss of teamness as a centrifugal force of global software teams is covered, the characteristics of a team must be presented. Carmel state that a real team must satisfy the following characteristics [29]:

A real team:

- is perceived to be a team by its members
- is recognized as a team by non-members
- has collective responsibility for its products
- shares responsibility for managing its work
- has a common goal or set of tasks
- works together on tasks that are interdependent
- demands peak performance from all members
- shares its rewards
- is small in number of members

It is clear from the presented characteristics that in good teams, team members help and complement each other and know the other team members well. In such team there is also high cohesion which leads to enhanced motivation, enhanced moral, greater productivity, harder work, more open communication and higher job-satisfaction [29]. The loss of teamness which occurs in global software teams, is a consequence of distance between team members, cultural differences and the loss of communication richness. Due to this loss of teamness global software teams satisfy less of these properties than collocated teams.

Loss of cohesion

One of the most important factors for a successful software development team is cohesion. Teams with high cohesion have many benefits as mentioned before. However, cohesion is more difficult for cross-cultural teams because of physical separation and lack of informal contact [29, 72, 70]. Due to these restrictions, team members may not be aware of the details of the work activities of their other team members, because they are only interested in their own activities. If awareness of current work is not completely spread across the whole team, misunderstandings can continue unnoticed and code conflicts can arise [4]. At the same time this lack of familiarity with remotely located colleagues can result in a lack of teamness and a reduced sense of trust [3].

Reduced trust

In a globally distributed setting trust is far harder to acquire or maintain than in classical development, because of the lack of close interpersonal contact [123, 105]. Pyysiäinen defines six problem areas in trust building, namely [123]:

- *Personal dispositions*
Personalities and interaction styles of parties in different companies can be ambiguous. It is possible that receivers interpreted a short and direct message, which was sent by a person who preferred messages that went straight to the point, as a commanding message and that the sender was not satisfied with their work. This misunderstanding is a consequence of different personal ways of expressing themselves; this could lead to misunderstandings and can reduce the level of trust.
- *Common history*
Because of the temporary nature of software development teams, companies often forgot to discuss the available documentation and whom to contact in specific issues. As a consequence of the lack of information and the fact that clear organizational charts and face-to-face meetings were lacking, background information was not sufficient exchanged. People did not get to know the roles, responsibilities and skills of each other and hesitated to spontaneously give and ask for help. This lack of knowledge can lead to a decreased level of trust and motivation.
- *Mediating third parties*
In the case of distributed development spontaneous transfer of knowledge via third parties was often blocked, because no mediating link persons between companies were available. The two companies did not know the exact reasons for delay in deliveries and testing, they did not know the causes for changes and some troubling bugs remained unclear. This kind of uncertainty about the positive intentions and motives of the other party can result in distrust.
- *Shared category membership*
One obvious problem is that people from different companies not actually feel that they are working towards a common goal. Reaching the sub-goals of each site seems

to have the highest priority instead of reaching the common goal. A lot uncertainty exist whether information is confidential and must be withheld from other companies or that the information is free accessible for other companies. When an employee is in doubt, the information is kept internal and is not accessible by the other companies. In this case ideas that would have helped the progress of the project are not exchanged. Another reason why it is difficult for a single site to identify them with a common goal is the limited feedback they receive on the quality of their work and that they could not perceive how their contributions are affecting the total progress. As a consequence, the commitment of the subcontractor can be weakened or even totally collapsed.

- *Predictable role behavior*

Co-located organizations often indicate roles in their processes but when an organization is operating in a global setting these clear prediction of the behavior of other people on the basis of their role is not possible because there was no such indication for the global organization. As a consequence it is hard to determine who has the right to decide on issues, especially at lower levels in the organization. It is even possible that a developer makes major changes to core modules, even though those kinds of tasks were not ascribed to him. These uncertainties have a negative influence of the level of trust between colleagues.

- *Internalized common rules*

It is often the case that basic issues as common terms are not clearly stated. There is also a lack of binding principles, communication and change-request protocols to be used in the development process. These unclear thresholds for changes can cause unnecessary and overlapping changes, contributes to uncertainties and reduces the level of trust between the different sites.

Due to the lack of close interpersonal contact and the problems in building trust, colleagues in a distributed setting are less inclined to help remote colleagues when workloads are heavy [73]. As a consequence developers may be doubtful of the knowledge, capabilities and skills of the team members from other sites [11, 131]. This impression may have a significant influence on the collaboration in a team.

Perceived threat from low-cost alternatives

Employees in the higher-cost economies can have the feeling that their jobs are most likely to be taken by their colleagues in lower-cost economies, creating a "*we versus they*" mentality [50, 74, 32]. As a result, they may not want to cooperate with their remote colleagues, which negatively affect the team work.

Increased Team size

Global teams in multiple sites are generally larger per task than co-located teams [29]. When a team becomes larger there are more employees involved which all have another role in the development process, this makes it more difficult for a manager to control the project

[53]. The main advantage of small teams is that it ensures effective communication among all team members [26]. This can also result in the increased level of trust and cohesiveness between team members. When a team becomes larger these advantages disappear and the success rate of the project drops down sharply [29, 26].

4.5 Cultural differences

Cultural differences can lead to misunderstandings which slow down the software development process [29]. These misunderstandings can arise due to excessive stereotyping, more in-group conversation and lower interpersonal attractiveness. In a global setting team members are part of multiple cultures, namely national cultures, corporate cultures, professional cultures, functional cultures and team cultures [29]. In section 4.5.1 the cultural fundamentals identified by Hofstede and Hall are presented and in section 4.5.2 the different subcultures and their accompanying challenges are discussed.

4.5.1 Fundamentals

Because there is no general theory of cultural differences, we use the dimensions identified by Hofstede [81] and Hall [67] in order to make the fundamentals of culture more tangible. Hofstede identified the following five dimensions of national culture [81]:

- *Power Distance Index (PDI)*
This fundamental dimension of culture has to do with how people think about equality and relationships with superiors and subordinates. In some cultures people are careful about expressing their opinion to superiors and show proper respect to their boss. On the other hand, superiors are expecting that subordinates only give input when their opinion is asked. Other cultures do not stick close to organizational hierarchy and managers even expect feedback from subordinates.
- *Individualism (IDV)*
Another dimension Hofstede distinguishes is the extent to which a person sees herself as an individual rather than part of a group. When people are expected to have their own opinion, are concerned with personal achievement, with individual rights and independence they are part of an individualistic culture. However, when people see themselves as part of a group they attach more importance to group welfare. This is the case in collectivist cultures.
- *Masculinity (MAS)*
This dimension reviews the differences in working culture; in a "*taking care of business culture*" the company counts above all. In this culture most of the decisions are taken from the organizational viewpoint. Employees, in this culture, are judged at dimensions such as competitiveness, assertiveness, promotions and bonuses. At the other extreme there are cultures in which the quality of life of an employee is viewed as more important. In these cultures decisions are taken from a more personal viewpoint.

- *Uncertainty Avoidance Index (UAI)*

The UAI indicates to what extent a culture programs its members to feel either uncomfortable or comfortable in unstructured situations. Unstructured situations are novel, unknown, surprising, ambiguous, different from usual. Uncertainty avoiding cultures try to minimize the possibility of such situations by strict laws and rules, safety and security measures. The key fundamental of these cultures is stability instead of innovation and change. At the opposite, Uncertainty accepting cultures are more tolerant of opinions different from what they are used to, and try to have as few rules as possible.

- *Long-Term Orientation (LTO)*

This is the last dimension identified by Hofstede and has to do with the relative importance of the short-term versus the long-term. Values associated with Long Term Orientation are thrift, persistence, diligence and patience. Characteristics of the Short Term Orientation are respect for tradition, fulfilling social obligations and protecting one's face. This dimension introduces both positive and negative values which can result into different challenges.

Edward Hall also introduced five dimensions of culture [67]:

- *Space*

An less more obvious cultural fundamental is space, there can be major differences between cultures because in one culture it is normal to stand one feet from each other while in another culture this is too close and is experienced as being rude. Another example is the seating arrangement at a table, one culture can have strict protocols of arranging the people while another culture has not. So, different cultures vary in their attitudes toward space.

- *Material goods*

The importance of material goods can also differ between cultures. In all cultures material goods are used for power and status, but the materials which provide power and status differ. In one culture a big office at the corner of the building indicates a high level of power and status while in another culture this is the case with an open office space next to his subordinates.

- *Friendship*

In cultures where it takes a long time to develop friendships, it is commonly the case that they are durable and involve a strong sense of mutual obligation. In these cultures people prefer to do business with people with whom they have developed a relationship. In other cultures, people make friends quickly and they do business with anyone who is capable.

- *Time*

There are two different time cultures; the linear time culture and the expandable time culture. In a linear time culture time and deadlines are taken seriously; people plan

processes in great detail, people treat deadlines very seriously and people are punctual in order to achieve time commitments. Expandable time cultures consider time commitments to be achieved only if possible. People change plans often and easily because delays are less important than the overall quality of the process. These differences in time perception can lead to misunderstandings.

- *Agreement*

Expressing agreement and disagreement varies by culture. In some cultures the detailed written contract is essential to agreement, while in other cultures a handshake is sufficient. In some cultures disagreement is openly and quickly expressed, while in other cultures open confrontation must be avoided. These differences must be handled in order to prevent misunderstandings.

A manager should have a high level of awareness of the above presented fundamentals in order to overcome the challenges which arise in a global organization [29].

4.5.2 Challenges

At this point, there is a clear understanding of the cultural fundamentals and we are able to identify the cultural challenges. These challenges are divided into five subcultures:

National Culture

National culture covers many facets of the daily life including, spoken language, national traditions, ethnic values and norms of behavior [29, 30].

Differences in language

In globally distributed development, some or all of the developers speak English, either as first or second language. Having to communicate in real-time can be overwhelming for these people finding it hard to follow the conversation [74, 90, 3]. This problem can even arise when the whole team exists of native speakers due to different dialects and local accents. This is the reason why most developers prefer to make use of Asynchronous communication tools because these allows non-native speakers to clearly formulate their position and makes it possible to check that they really make their point clear before sending the message [4].

Differences in ethical values

Next to differences in language between the various cultures, there are also differences in the perception of space and material goods which play a major role in GSD. In order to exploit the benefits of GSD these challenges must be solved.

Corporate Culture

Organizational culture encompasses the corporate norms and values of an organization, the organization can be small and co-located but it can also be a large and globally dispersed multinational [30]. Organizational culture includes, management styles, appraisals, rewards

and communication styles used by the employees [29].

Differences in organizational vision

The cultural fundamentals *taking care of business*, *Long-Term orientation* and *risk avoidance* have a great impact on the management of a multi-sited organization. In some cultures the company counts above all, while in other cultures the quality of life is viewed as more important. Between the members of other cultures there can also be a difference in attitude towards risk avoidance as well as there can be a difference between the relative importance of the short-term and the long-term. When there is no attention given to these inconsistencies, the different sites can resolve management issues in a different way. Due to these different approaches a lack of understanding can arise between the multiple sites of an organization [29, 120].

Differences in managing individualism and collectivism

When an organization has multiple sites across the world there are multiple cultures involved in the organization. One of challenges which arise is the combination of individualistic cultures and collectivist cultures. In an individualistic culture individuals are rewarded for their achievements this can lead to embarrassment and loss of team harmony for team members with a collectivist cultures. It is also possible that a collectivist, who expects to be told honestly what is the matter, is insulted because members of a collectivist culture do not confront him with issues directly, but handle these in a group fashion. So, a meeting between members of individualist and collectivist cultures can result in mutual conflicts just because of their styles of communication [29].

Differences in terms of agreement

Another challenge in the area of corporate culture is the different way cultures use to express agreement and disagreement. In some cultures a detailed written contract is essential while in others a handshake is sufficient. These can lead to misunderstandings by both parties since one could believe that he had an agreement while the other was not aware of this [29].

Differences in time perception

Different time perceptions can lead to misunderstandings. A linear time person finds it impolite or even unacceptable when someone is not able to achieve time commitments, and may even conclude that he cannot count on this person. However, someone who sees time as expandable is more concerned with the other factors of the process, such as quality. These conflicting ways of thinking can lead to a lack of understanding between members of these two cultures [29, 74].

Professional Culture

The professional culture is ingrained in us through highly structured formal education during many years. This culture is maintained by training programs and by taking courses. A professional culture is strong, since a person often chooses his profession for life [29].

Differences in Quality Assessment

There is a great difference in the quality assessment of western cultures and the quality assessment of other cultures. These have to do with the different ways of education, for instance, in Japan the focus is on recording and fixing of all the existing bugs of a system. They do not care much about the criticality of the bugs; both the critical and minor bugs must be fixed before the newest version of the product is released. In the western culture the focus is more on the critical bugs, if these are resolved a new version of the product can be released [29]. Minor bugs are solved in a next release.

Differences in design

Another challenge which arises due to different formal education is the difference in approaching development problems. In the design phase western software architects approach the problem from a *top-down approach*. First the global design is made and after that the details are filled in. Software architects from other cultures can begin with many details which in later phases emerge into the big picture. This approach is called the *bottom-up approach*. Due to these differences software architects from both cultures can be frustrated during the design sessions [29, 120].

Functional Culture

A functional culture is made up of the norms and habits associated with functional roles within the organization. There exist different functional roles within the organization like, marketing, sales, finance, R&D and manufacturing [29].

Differences in attitude toward hierarchy

In organizations which operate on a global platform, having employees from a range of different cultures, problems may arise because of different attitudes toward hierarchy [29, 74]. In cultures that do not revere hierarchy it is commonly accepted that employees express their opinion to superiors and have discussions about the development process. In other cultures, that do revere to hierarchy, these interactions are less likely to occur and can even been seen as impolite. These differences should be managed adequately in order to prevent misunderstandings [29].

4.6 Overview

All the challenges raised above are summarized in table 4.1. In this table the challenges which can be categorized into more than one problem area have several plusses on a row; a double plus indicates the main problem area, the section in which the challenge is discussed; a single plus indicates that there is a relation between the challenge and the corresponding problem area; and an empty cell indicates that there is no relation between the problem area and the challenge at hand. In this fashion, table 4.1 gives an effective and structured overview of the many challenges involved with global software development and their accompanying problem area(s). We can, for example, see that the increased effort it takes

4. THE CHALLENGES OF GLOBAL SOFTWARE DEVELOPMENT

to initiate contact with colleagues at another location has high impact on the frequency of communication, represented by a double plus in the table. Cultural differences between the members of a team and the lack of teamness aggravate this challenge; this relation is indicated by a single plus in the table.

Challenge	GD	CCB	LCR	LT	CD
Lack of informal communication	++	+	+	+	+
Increased effort to initiate contact	++			+	+
Reduced hours of collaboration	+	++		+	
Lack of shared understanding	+	++	+	+	
Increased dependency on technology		++			
Increased complexity of the technical infrastructure		++			
Communication delay	+	+	++	+	
Loss of cohesion	+		+	++	+
Reduced trust	+		+	++	+
Perceived threat from low-cost alternatives				++	
Increased team size	+	+		++	
Differences in language					++
Differences in ethical values					++
Differences in organizational vision					++
Differences in managing individualism and collectivism					++
Differences in terms of agreement					++
Differences in time perception					++
Differences in quality assessment					++
Differences in design					++

GD	Geographic dispersion
CCB	Control and coordination breakdown
LCR	Loss of communication richness
LT	Loss of teamness
CD	Cultural differences

Table 4.1: The challenges of GSD and their classification

Chapter 5

Non technological support for alleviating distance in global software development

Managers in a global distributed organization are experimenting and quickly adjusting their tactical approaches in order to take optimal advantage of the benefits offered by GSD. However, the most intuitive approach for alleviating distance is to apply technological solutions, these solutions are discussed in chapter 8. In this chapter the focus is on non-technological support which reduces the problems of distance in global software development. These non technological solutions are aimed at reducing intensive collaboration between team members, the impact of national and organizational cultural differences and temporal distance.

5.1 Reduce intensive collaboration

The transitions of tasks between the *Center* and the *Foreign Entity* is one of the main organizational difficulties of dispersed organizations [30]. Usually the *Center* is a firm in either North America or in the European Union. The *Foreign Entity* is usually located in a newly industrialized or developing nation [30]. The tasks which need to be divided between these two locations range from well defined and structured to poorly defined and unstructured. Unstructured and hard to define tasks are intuitively associated with an increased level of coordination complexity between the Center and the Foreign Entity. Figure 5.1 shows that this is not always the case and that organizations can move to the far left or to the far right, in order to reduce the coordination complexity. Organizations which move to the lower left corner in figure 5.1 are outsourcing relatively straightforward tasks with low complexity such as maintenance activities, help desks and data centers to their foreign entity. These tasks are more manageable over distance, because the need to communicate and clarify requirements is relatively limited and the tasks are relatively stable [64, 30, 104]. On the other hand, organizations are moving to the lower right corner, these organizations are outsourcing tasks which are relatively complex and unstructured to their foreign entity. The foreign entity takes full responsibility for a system, product or corporate process. This alleviates many of the distance problems because the foreign entity is not using links with

5. NON TECHNOLOGICAL SUPPORT FOR ALLEVIATING DISTANCE IN GLOBAL SOFTWARE DEVELOPMENT

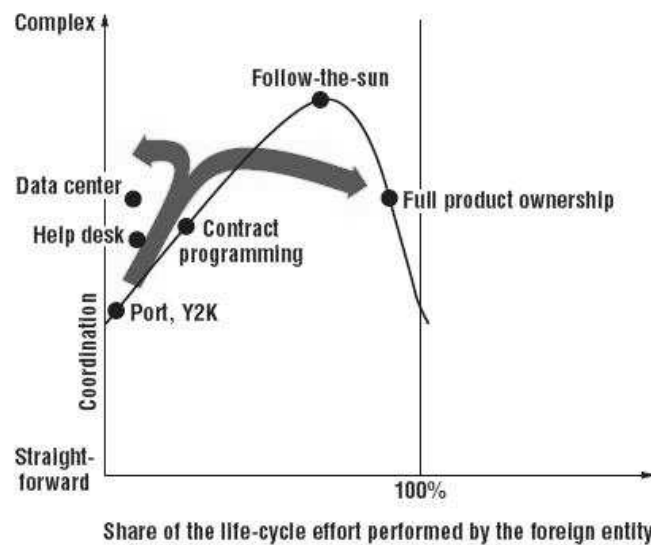


Figure 5.1: Alternative paths to alleviating intensive collaboration [30]

the center as frequently [30, 104]. Finally, figure 5.1 shows that organizations which are using the follow-the-sun approach are characterized by a very dense web of coordination that is needed to transfer knowledge and collaborate on tasks. In this case it is not possible to move to the far left or to the far right because each site must be able to add its own value to the process so that the other site can quickly proceed to add its own value without further clarifications.

5.2 Reduce cultural distance

Cultural distance stems from the degree of difference between the center and the foreign entity. In figure 5.2 the structural arrangements for global software development are plotted along the distance from center's national culture and the distance from center's organizational culture. The upper left corner represents organizations with negligible cultural differences due to cultural unity while the lower right corner represents organizations with substantial cultural differences due to the cultural diversity. Carmel et al. presented four approaches, represented by the arrows in figure 5.2, in order to alleviate the cultural distance between the different sites [30]:

1. *Bridgehead*

The first of these arrangements is the offshore-onshore bridgehead, which reduces both national and organizational cultural distance. This arrangement is also known as the 75/25 rule, essentially 75 percent of personnel work occurs offshore, while 25 percent occurs onshore. The individuals assigned to work onshore are more experienced and have a corresponding cultural notion, both to colleagues and customers.

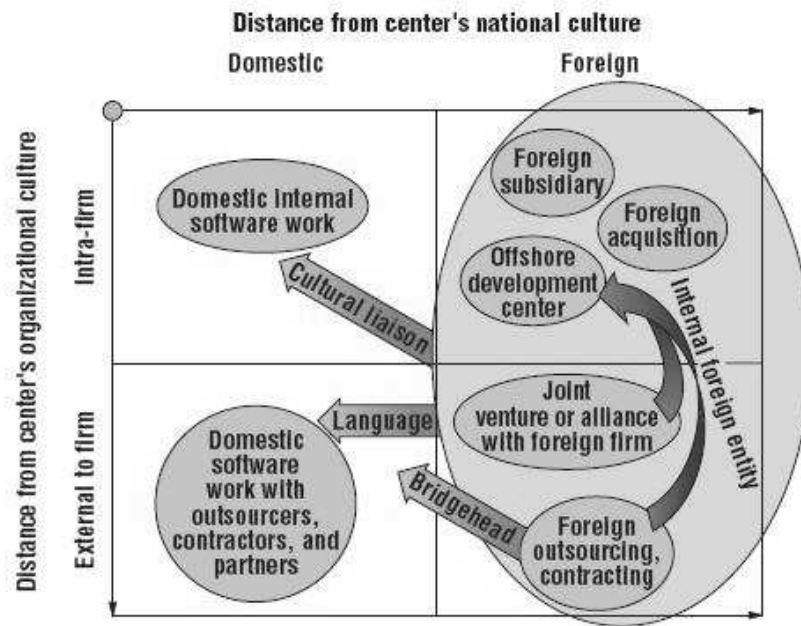


Figure 5.2: A taxonomy of structural arrangements for software development [30]

This results in less misunderstandings and a higher understanding of customer requirements. In other words the 25 percent of personnel that are onshore effectively serve as a bridge between the customer and the offshore workforce in order to reduce cultural distance.

2. *Internalization of Foreign Entity*

In order to reduce organizational distance some American and European companies are opening internal-to-the-firm foreign software centers. By internalizing global software development, and avoiding collaboration with external foreign partners, the distance in organizational culture is reduced.

3. *The cultural liaison*

The informal role of the cultural liaison is to facilitate the cultural, linguistic and organizational flow of communication and to bridge cultures, mediate conflicts, and resolve cultural miscommunications. This role might be fulfilled by an individual who travels back and forth between the key stakeholder sites [83]. As a consequence both the organizational as the national differences in culture are reduced.

4. *Language*

The last approach presented by Carmel et al. is the influence of language; spoken language is an important component of national cultural distance. Many decision-makers hesitate to engage in international alliances, especially with nations in which the command of English is weak. This language factor is one of the reasons for the

success of offshore IT work in countries with strong English language capabilities. Some organizations invest in English as a foreign language course in order to improve professional communication.

5.3 Reduce temporal distance

Despite of the considerable power asynchronous technologies for dispersed tasks synchronous communication is still preferred. Advantages of synchronous communication include resolving miscommunications, misunderstandings and small problems before they become unmanageable [30]. When two teams use asynchronous communication techniques a small issue can take days, but a brief conversation can quickly clarify the problem. Another disadvantage of being involved in global work is the need to compromise personal life to speak colleagues in other time zones. The goal of this approach is to minimize the time-zone differences to be able to use effective synchronous communication [30, 83]. However, reducing temporal distance eliminates the advantage of the follow-the-sun approach.

Part III

Extending Global Software Development with aspects of the agile development process

Chapter 6

Agile methodologies

Agile software development methods are an approach to software development in which the process of software development is much more flexible than with traditional development methods. In this chapter we will attempt to define what an agile method is and discuss the reasons behind the current uprise. After this, an overview of both Scrum and eXtreme programming will be given followed by a less in depth overview of Crystal methodologies, Adaptive software development and Feature driven development. In conclusion we will link agile methodologies with global software development by defining aspects of agile software development and by discussing which of those aspects have an impact on the distances caused by working globally distributed.

6.1 What are agile methodologies?

The software engineering discipline has existed ever since the 1960s and has come a long way since then [95]. First, software was written without much of a plan and the design of the system was determined from many short term decisions. This only worked well for small systems, since for larger systems fixing bugs and adding new features was too hard [8]. So, disciplined processes, called methodologies [56], were imposed upon software development, with the aim of making software development more predictable and more efficient [8]. Traditional, plan-driven methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing and deployment [8]. They start with determining the set of requirements as complete as possible. Then based on these requirements a plan of development is formulated [141]. Characteristics of these traditional, heavyweight methodologies are: [8]

- *Predictive approach*
Concerns following a predictive and repeatable development approach in which a large part of the process is planned in advance.
- *Comprehensive Documentation*
Concerns an extensive and explicit specification of all information in the project, with the requirements being the most important example.

- *Process Oriented*
Concerns having a well defined procedure for the tasks performed by all project members.
- *Tool Oriented*
Concerns that Specific tools must be in use for completion and delivery of each task.

A few Well-known examples of traditional methods are:

- *The waterfall method [128]*
This method defines phases of which each project consists and a structured progression between them. Each of these phases in turn consists of a predefined set of activities and deliverables.
- *The spiral model [22]*
This method combines the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It combines elements of both design and prototyping-in-stages, in an effort to combine advantages of top-down and bottom-up concepts [8]. This development approach is much more flexible than the waterfall method, it however still needs to be planned methodically, with tasks and deliverables identified in each step of the model [69].
- *The Unified Process [85]*
This method aims at working with short, time-boxed iterations, accept changes readily, and reflect team collaboration. But at the same time it follows four general phases in a linear fashion and as the development proceeds less time is spent on requirements and analysis, and more time is spent on construction, testing and transition.

Currently, many sources report the development of lightweight methodologies as a reaction to the inflexibility of these existing, heavyweight, methods [56, 80, 1, 87, 41, 141, 8, 69]. Examples of these new methodologies are:

- *Scrum [132, 133]*
- *eXtreme programming [13, 12, 62, 15]*
- *Crystal methods [38, 39]*
- *Adaptive software development [77, 79, 80]*
- *feature driven development [37, 110]*

After the development of these new methodologies, in 2001, seventeen prominent process methodologists held a meeting to discuss future trends in software development. They noticed their methods had many commonalities and defined a name for methods of this kind: "agile methods". They formed the "Agile Alliance" and wrote "The agile manifesto" [16] in which they defined four core values:

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

They also defined a set of principles in which these values are realized:

- AM₁ Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- AM₂ Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- AM₃ Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- AM₄ Business people and developers must work together daily throughout the project.*
- AM₅ Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- AM₆ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- AM₇ Working software is the primary measure of progress.*
- AM₈ Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- AM₉ Continuous attention to technical excellence and good design enhances agility.*
- AM₁₀ Simplicity—the art of maximizing the amount of work not done—is essential.*
- AM₁₁ The best architectures, requirements, and designs emerge from self-organizing teams.*
- AM₁₂ At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

From these values and principles it can be gathered that the agile methods claim to place more emphasis on people, interaction, working software, customer collaboration, and change, rather than on processes, tools, contracts and plans. The main difference between agile methods and traditional method which are iterative, like the spiral model and the unified process mentioned earlier, is that agile methods are also responsive and flexible *during* the iterations, while the iterative traditional methods are merely flexible between subsequent iterations [132]. Whether this approach actually works remains to be seen, as empirical studies evaluating the effectiveness and the possibilities of using agile software development

methods are still scarce [8]. Awad [8] performed a survey among software practitioners in government and commercial organizations in Perth. The most noticeable results from this study are that practitioners feel that agile methodologies lower development costs in small-scale projects, have no effect on cost in medium-scale projects and have a negative effect on large-scale projects. Cockburn et al. confirm that agile development is more difficult in larger teams [40]. They however also state that it is possible: *Nevertheless, it is interesting to occasionally find successful agile projects with 120 or even 250 people* [40]. Another result from the survey written by Awad was that the surveyed practitioner, on average, felt that agile methods were harmful for the quality of the final product. Shine technologies performed a similar survey [137] which produced quite different results. Their results showed nothing but positivism towards agile methods. Highsmith and Wysocki [78] also performed a survey, but they focused on researching software development processes used amongst organizations and organizational agility. One of the results of this survey was that agile methodologies are indeed very popular the moment. A final study we will mention here is a survey by Salo and Abrahamsson [129]. This survey aimed at gathering information with respect to the application of two agile methods, namely Extreme Programming and Scrum, in a number of European organizations of embedded software known to be interested and active in experimenting with agile software development methods. The results indicate embedded software development organizations seem to be able to apply agile methodologies and that the appreciation of these methodologies seems to increase once they are adopted and applied in practice.

This section will be concluded with two summaries of the differences between heavyweight and lightweight methodologies. Figure 6.1 shows the summary by Awad [8] and figure 6.2 shows the summary by Boehm [23]. In the remaining sections of this chapter, the agile methodologies mentioned in this section will be discussed.

	Agile Methods	Heavy Methods
Approach	Adaptive	Predictive
Success Measurement	Business Value	Conformation to plan
Project size	Small	Large
Management Style	Decentralized	Autocratic
Perspective to Change	Change Adaptability	Change Sustainability
Culture	Leadership-Collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable/Exploratory	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in Project	End of Project
Team Size	Small/Creative	Large

Figure 6.1: Difference in Agile and Heavyweight Methodologies [8]

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

Figure 6.2: Difference in Agile and Heavyweight Methodologies [23]

6.2 Scrum

Scrum [132] is an agile process that emphasizes a set of project management values and practices [93]. It does not define any specific software development techniques for the implementation phase. Scrum concentrates on how the team members should function in order to produce the system flexibly in a constantly changing environment [1]. The term '*scrum*' originated from a strategy in the game of rugby [132] where it denotes a way of restarting the game, either after an accidental infringement or when the ball has gone out of play. In a scrum the players of both teams stand opposite to each other, in a squatted position, with their heads interlocked between the heads of the most forward players of the opposite team. A player of the team that did not cause the scrum (did not infringe) throws the ball into the scrum and then the players of both teams try to compete for the ball by trying to hook the ball backwards with their feet. In doing this teamwork is very important since two teams are trying to push each other backwards.

6.2.1 Artifacts

The two most important artifacts during the Scrum process are:

- *Product Backlog*
The evolving, prioritized, queue of product functionality requirements that are not yet adequately addressed by the current version of the system. Bugs, defects, customer requested enhancements, competitive product functionality, competitive edge functionality, and technology upgrades are backlog items [132, 133].
- *Sprint Backlog*
The portion of the product backlog, scheduled for the current iteration, called a sprint. The items of the sprint backlog are broken down into tasks. For each of these tasks the following is maintained [141]:
 - task description
 - task originator
 - task owned
 - status

- time remaining until completion

Each day the time remaining for items is re-estimated to reflect the current knowledge regarding the item.

6.2.2 Roles and responsibilities

The roles which can be identified in a Scrum development team are [141]:

- *Product Owner*
Is responsible for creating and prioritizing the Product Backlog, choosing what will be included in the next iteration/Sprint, and reviewing the system (with other stakeholders) at the end of the Sprint.
- *Scrum Master*
Measures progress, removes obstacles, and leads the team meetings, but at the same time participates in product development.
- *Scrum team member*
Scrum teams are self-directed and self-organizing teams. The team commits to a defined goal for an iteration and is given the authority, autonomy, and responsibility to decide how best to meet it [141].

6.2.3 Process

The Scrum process, depicted in figure 6.3, consists of three phases: pre-game, game and post-game. The process begins with the pre-game phase which is an explicitly defined process with well-defined inputs and outputs [132]. In this phase a definition of a new release based on currently known backlog is made, along with an estimate of its schedule and cost. Subsequently, a high-level architectural design of the implementation of the backlog is made.

The next phase is the game phase and is an empirical process. This means that many of the processes in the phase are unidentified or uncontrolled. To handle this uncertainty the phase consists of a number of iterations called sprints. A Sprint is a time-boxed period of time of usually 30 working days [141, 8, 69]. During a sprint new release functionality, selected for this sprint, is developed, while constantly adjusting with respect to time, requirements, quality, cost, and competition. If explicit process knowledge is available it is used, otherwise tacit knowledge and trial and error is used to build process knowledge. The sprint cycle is depicted in figure 6.4. Between sprints, when one sprint finished and the next one begins, controls, including risk management, are defined for the next sprint in so-called "Sprint Planning meetings". This is done to avoid chaos while maximizing flexibility [132].

The scrum process ends with the end-game phase. This phase, like the pre-game phase, is an explicitly defined process with well-defined inputs and outputs. In this phase the preparations for release are made, including final documentation, pre-release staged testing, acceptance testing, evaluation and the actual release.

Scrum Methodology

■ Pregame

- Planning
- System Architecture/High Level Design

■ Game

- Sprints (Concurrent Engineering)
- Develop (Analysis, Design, Develop)
- Wrap
- Review
- Adjust

■ Postgame

- Closure

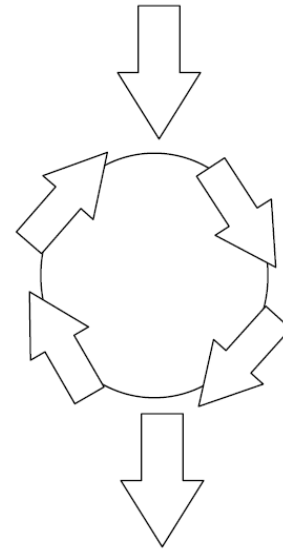


Figure 6.3: The general phases of Scrum [132]

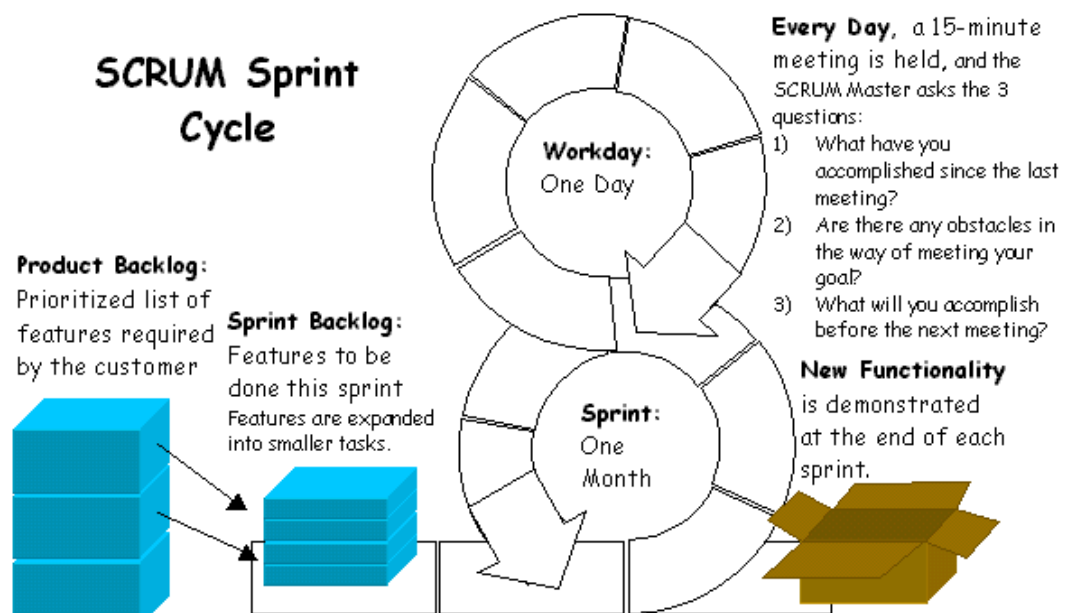


Figure 6.4: The Scrum sprint cycle [36]

6.3 eXtreme Programming

Extreme programming (XP) is an agile process methodology which can be characterized by short development cycles, incremental planning, continuous feedback, reliance on communication, and evolutionary design. The individual pieces of XP are not new, but XP has integrated them in such a way as to form a new methodology. The term '*extreme*' comes from taking principles and practices, which to Beck seem common-sense, to extreme levels [15]. It is based upon five underlying values which XP's practices attempt to elicit. The five core values are [141]:

- *Communication*
Good communication helps reduce misunderstandings and increase cooperation in teams. Beck states: "*Problems with projects can invariably be traced back to somebody not talking to somebody else about something important*" [12].
- *Simplicity*
By just designing what is explicitly asked for by the customer in the requirements, the chance of making something the customer does not want decreases.
- *Feedback*
By acquiring timely feedback, misunderstandings in specification are discovered more quickly.
- *Courage*
The development team needs to have courage in its actions and decision making to perform the right actions and make the right decisions.
- *Respect*
Since XP, and in fact most agile methodologies, revolve around close collaboration in teams, it is important that teams get along well.

The initial version of the XP software methodology [12] published in 2000 had 12 programmer-centric, technical practices. In 2005, XP was changed to include 13 primary practices and 11 corollary practices [15]. The primary practices are intended to be useful independent of each other and the other practices used, though the interactions between the practices may amplify their effect. The corollary practices are likely to be difficult without first mastering a core set of the primary practices [141].

The 13 primary technical practices of XP are: (taken from Williams [141]):

XP₁ Sit together

The whole team develops in one open space.

XP₂ Whole team

Utilize a cross-functional team of all those necessary for the product to succeed.

XP₃ Informative workspace

Place visible wall graphs around the workspace so that team members (or other interested observers) can get a general idea of how the project is going.

XP₄ Energized work

XP teams do not work excessive overtime for long periods of time. The motivation behind this practice is to keep the code of high quality (tired programmers inject more defects) and the programmers happy (to reduce employee turnover).

XP₅ Pair programming

Refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.

XP₆ Stories

The team write short statements of customer-visible functionality desired in the product. The developers estimate the story; the customer prioritizes the story.

XP₇ Weekly cycle

At the beginning of each week a meeting is held to review progress to date, have the customer pick a weeks worth of stories to implement that week (based upon developer estimates and their own priority), and to break the stories into tasks to be completed that week. By the end of the week, acceptance test cases for the chosen stories should be running for demonstration to the customer to drive the next weekly cycle.

XP₈ Quarterly cycle

The whole team should pick a theme or themes of stories for a quarters worth of stories. Themes help the team reflect on the bigger picture. At the end of the quarter, deliver this business value.

XP₉ Slack

In every iteration, plan some lower-priority tasks that can be dropped if the team gets behind such that the customer will still be delivered their most important functionality.

XP₁₀ Ten-minute build

Structure the project and its associated tests such that the whole system can be built and all the tests can be run in ten minutes so that the system will be built and the tests will be run often.

XP₁₁ Test-first programming

All stories have at least one acceptance test, preferably automated. When the acceptance test(s) for a user story all pass, the story is considered to be fulfilled. Additionally, automated unit tests are incrementally written using the test-driven development (TDD) [14] practice in which code and automated unit tests are alternately and incrementally written on a minute-by-minute basis.

XP₁₂ Continuous integration

Programmers check in to the code base completed code and its associated tests several

times a day. Code may only be checked in if all its associated unit tests and all of unit tests of the entire code base pass.

XP₁₃ Incremental design

Rather than develop an anticipatory detailed design prior to implementation, invest in the design of the system every day in light of the experience of the past. The viability and prudence of anticipatory design has changed dramatically in our volatile business environment. Refactoring to improve the design of previously-written code is essential. Teams with robust unit tests can safely experiment with refactorings because a safety net is in place.

The 11 collary technical practices of XP are: (taken from Williams [141]):

XP₁₄ Real customer involvement

The customer is available to clarify requirements questions, is a subject matter expert, and is empowered to make decisions about the requirements and their priority. Additionally, the customer writes the acceptance tests.

XP₁₅ Incremental deployment

Gradually deploy functionality in a live environment to reduce the risk of a big deployment.

XP₁₆ Team continuity

Keep effective teams together.

XP₁₇ Shrinking team

As a team grows in capacity (due to experience), keep their workload constant but gradually reduce the size of the team.

XP₁₈ Root cause analysis

Examine the cause of a discovered defect by writing acceptance test(s) and unit test(s) to reveal the defect. Subsequently, examine why the defects was created but not caught in the development process.

XP₁₉ Shared code

Once code and its associated tests are checked into the code base, the code can be altered by any team member. This collective code ownership provides each team member with the feeling of owning the whole code base and prevents bottlenecks that might have been caused if the owner of a component was not available to make a necessary change.

XP₂₀ Code and tests

Maintain only the code and tests as permanent artifacts. Rely on social mechanisms to keep alive the important history of the project.

XP₂₁ Single Code Base

Make use of a single code base to avoid having to maintain various versions of the project.

XP₂₂ Daily deployment

Put new code into production every night.

XP₂₃ Negotiated scope contract

Fix the time, cost, and required quality of a project but call for an on-going negotiation of the scope of the project.

XP₂₄ Pay-per-use

Charge the user every time the system is used to obtain their feedback by their usage patterns.

6.3.1 Artifacts

In XP documentation is kept to a minimum and information is instead primarily shared by via oral communication, the code itself, and tacit knowledge transfer. The "documentation" that XP does use is usually of an unofficial, improvised nature, like paper index cards which contain brief requirements and visible wall graphs [141].

6.3.2 Roles and responsibilities

The roles which can be identified in a XP development team are [12, 1, 141]:

- *Manager*
Is responsible for the project as a whole. The manager forms the team, obtains resources, manages people and problems and interfaces with external groups.
- *Coach*
Guides the XP team members through the XP process and is responsible for this process. The coach is typically a programmer and not a manager.
- *Tracker*
Traces the estimates made by the team and gives feedback on their accurateness in order to improve future estimations. The tracker also tracks the progress of each iteration and whether this progress is sufficient to reach the specified goals within the given resource and time constraints. The tracker is a programmer, not a manager or customer.
- *Programmer*
Writes tests, design, and code; refactors; identifies and makes estimates for task duration.
- *Tester*
Helps the customer write functional tests, run the functional tests regularly, broadcast the test results and maintain the testing tools. The tester may also be a programmer.
- *Customer*
Determines the requirements and functional tests, as well as the priority of each requirement. The customer decides when a requirement is satisfied.

- *Consultant*

External member with specific technical knowledge needed at some time during the project. The consultant's expertise is used to solve specific problems.

6.3.3 Process

The XP process, depicted in figure 6.5, consists of the following six phases [12, 1]:

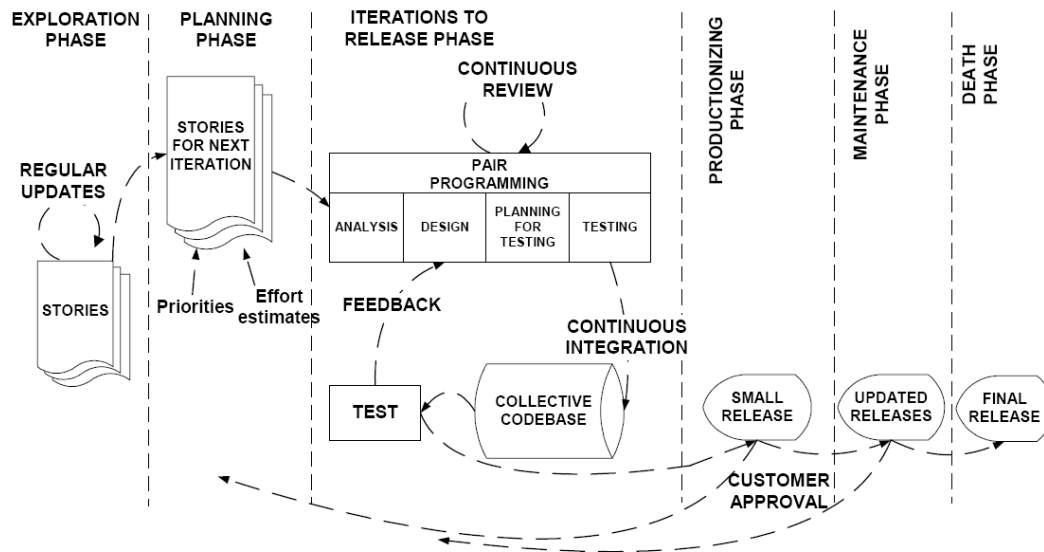


Figure 6.5: The life cycle of the XP process [1]

- *Exploration*

In this phase the customers specify the features they wish to be included in the first release of the system, while at the same time, the project team familiarizes itself with the tools, technology and practices they will be using in the project. This phase takes between a few weeks and a month, depending largely on how familiar the technology is to the programmers.

- *Planning*

In this phase the priority of the requirements is set and an agreement of the content of the first small release is made. This is done by estimating how much effort each requirement would take to implement. The time span of the first release does not normally exceed two months, while the planning phase itself takes a couple of days.

- *Iterations to release*

In this phase several iterations of the system will be gone through in order to arrive at the first release. The schedule determined in the planning phase is broken down to a number of iterations that will each take one through four weeks to implement. The

first iteration creates a system that possesses the architecture of the whole system. To do this, requirements which enforce building the structure of the whole system are selected. At the end of every iteration the functional tests defined by the customer are run. At the end of the final iteration the system is ready for production.

- *Productionizing*

In this phase extra testing and checking of the performance of the system is done, in order for the system to be able to be released to the customer. During this, required changes to the system may be found, and a decision is made whether they are carried out for the current release or documented for later implementation. During this phase, iterations may need to be shortened to one week.

- *Maintenance*

In this phase, the first release of the system is taken into production for customer use. This release must be maintained while at the same time producing new iterations. In order to do this, this phase also requires customer support tasks and may thus decelerate the development velocity or require the addition of new people into the team.

- *Death*

When all the requirements of the customer, including the ones regarding performance and reliability, are incorporated into the system, this phase is reached. In this phase the necessary documentation of the system is written, as no more changes to the architecture, design and code are made. Another reason for the death phase to occur is discontinuation of further development of the system.

6.4 Other agile methods

Crystal methodologies

Crystal methodologies are part of a family of methodologies called the Crystal family [39]. All of these methodologies provide guidelines of policy standards, work products, "local matters", tools, standards and roles to be followed in the development process [1]. With regard to the development process, only two commonalities exist among the entire crystal family: Incremental cycles may not exceed four months and reflection workshops must be held after every delivery so that the methodology is self-adapting [141].

The word "crystal" in the name of this methodology-family refers to the various facets of a gemstone, each one being a different viewpoint towards one and the same underlying core [80]. The underlying core represents values and principles, while each facet represents a specific set of elements such as techniques, roles, tools, and standards. An example of a value that is common among the entire crystal family is the importance of proper communication and collaboration among the team members [39]. The different methods are assigned colors arranged in ascending opacity, where a less opaque color represents a more agile methodology. The most Agile version is Crystal Clear, followed by Crystal Yellow,

Crystal Orange and Crystal Red. The version of crystal you use depends on the number of people involved, which translates into a different degree of emphasis on communication [41]. At the moment only crystal clear and crystal orange have been developed [1].

Adaptive software development

Adaptive Software Development (ASD) [77] is a process methodology focused on iterations and constant prototyping and is particularly suited for the development of large and complex software systems. The aim of ASD is to provide a framework with just enough guidance so the project does not fall into chaos, but not so much that it would suppress emergence and creativity [1].

According to HighSmith An iteration in the ASD process: *"needs to be short, so teams can learn from small rather than large mistakes"* [77]. Such a iteration is, as depicted in figure 6.6, is divided into three phases [77, 79]:

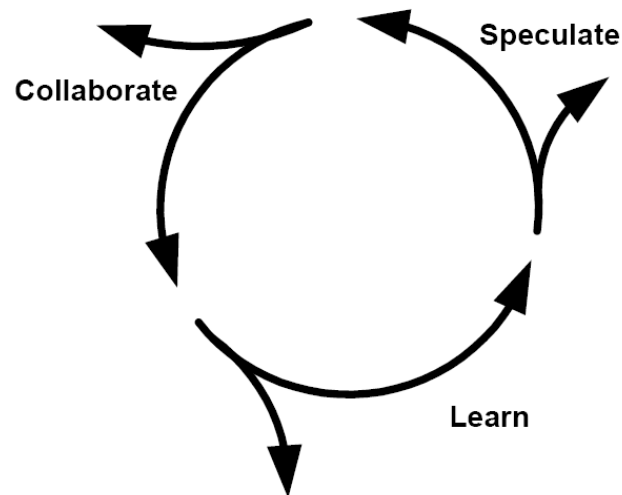


Figure 6.6: The ASD cycle [79]

- *speculate phase*

In this phase the objectives, vision, goals, and requirements of the system to be developed in the current iteration are specified. The name speculate is chosen instead of planning because planning seems to imply that uncertainty must be avoided.

- *Collaborate phase*

In this phase the system to be built for the current iteration is constructed.

- *Learn phase*

In this phase the products constructed during the collaboration phase are exposed to a variety of stakeholders to ascertain value. *"Customer focus groups, technical reviews, beta testing, and postmortems are all practices that expose results to scrutiny"* [77]. Feedback from this phase as well as the system constructed during this iteration are inputs for the next iteration, so it is possible to learn from mistakes made during this iteration.

Feature driven development

Feature Driven Development (FDD) [37, 110] is an agile and adaptive approach to develop systems, which does not cover the entire software development process, but just the design and building phases. The approach prescribes emphasis on quality aspects throughout the process, short iterations with frequent and tangible deliveries and accurate monitoring of other progress of the project [37, 110, 1]. The process, which is depicted in figure 6.7, consists of five sequential phases, of which the first three are done at the beginning of the project and the last two form the iterative part of the process:

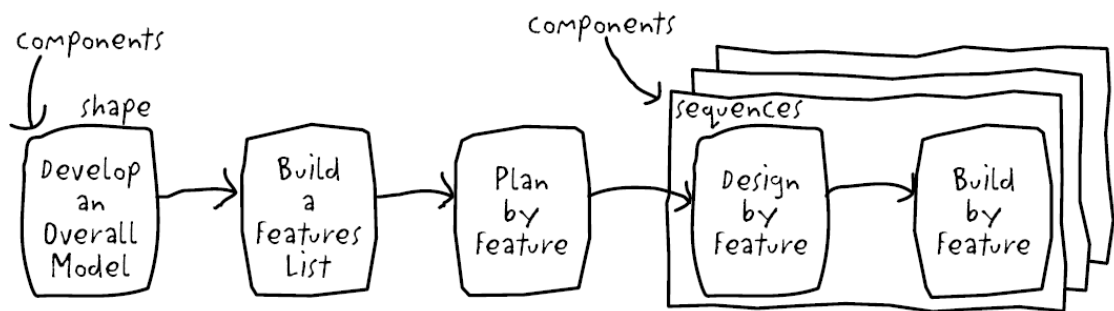


Figure 6.7: The five phases of FDD [37]

- *Develop an overall model*

When this phase begins, the domain experts already are aware of the scope, context and requirements of the system to be built. So, requirement gathering and optionally, the creation of the documentation of these requirements, has been done at this stage. In this stage all team members and the chief architect are informed of the high-level description of the system with so-called *"walkthroughs"* of the system, at various levels of detail.

- *Build a features list*

In this phase a categorized list of the features to support the requirements is produced. The size of these features is so that they can be implemented in ten days. Features requiring more time than ten days are broken down into sub-features. Finally the list of features produced, is reviewed by the users and sponsors of the system to test if the list is both valid and complete.

- *Plan by feature*

In this phase, the categories of features are sequenced with respect to their priority and dependencies among another. Then the categories are assigned to developers who will be responsible for the development of that category, and a schedule of milestones is created.

- *Design by feature and build by feature*

In these phases all the features are produced in an iterative manner. Each iteration, a set of features is selected and the developers responsible for the development of the feature categories selected in this iteration, form teams to perform the implementation. An iteration usually takes anywhere between a few days and two weeks. When an iteration is finished the completed features are integrated into the main build and a new set of features is selected for the next iteration.

6.5 Aspects of agile software development

In this chapter we introduced the concept of agile methodologies in the context of software engineering in general. This thesis, however, is concerned with the combination of agile methodologies and global software development. Therefore we will define a subdivision of agile software development into aspects to be able to systematically discuss how agile software development can be beneficial explicitly with respect to global software development. With the term "*aspect of agile software development*" we denote the goals agile software development attempts to accomplish. More explicitly, for the agile manifesto and agile methodologies which define explicit practices, these aspects can be seen as *the outcome of using an agile practice from a certain subset of agile practices*. In this section we will discuss these aspects. For each of these aspects we will start by giving a definition of what the aspect entails followed by a clarification of how exactly the aspect is reflected in the agile manifesto, and the eXtreme programming and Scrum agile methodologies, because these methodologies were most thoroughly examined in this chapter. This discussion will be most thorough and explicit with respect to the Agile Manifesto and XP since these define design practices explicitly. Finally we will discuss whether or not the aspect offers benefits exclusive to global software development. In chapter 2 we stated that all challenges associated with global software development originate from the increased distances of geographical, temporal and socio-cultural nature. Therefore this final discussion will entail whether or not the aspect under discussion is able to ease the dealing with at least one of the three distances. The division of the aspects into aspects that do not, or only minorly, influence distance and those that do, is made because our interest in this thesis mainly lies on using agile software development to explicitly improve global software development. Therefore aspects that offer similar benefits in a distributed environment as they do in a co-located environment are of less concern in this thesis. The relation between the aspects and the practices of the Agile Manifesto and eXtreme programming will be summarized in table 6.1 and 6.2 at the end of this section.

A₁ Close collaboration among the members of the development team**Short description**

This aspect concerns that all members of the development team should work together daily throughout the project in a very close way, while communicating frequently.

Relation to the agile methodologies

This aspect is reflected in the agile manifesto in the following way. For one it urges that business people and developers should work together daily throughout the project (*AM₄*). Also it states that the best architectures, requirements and designs emerge from self-organizing teams (*AM₁₁*). For teams to be self organizing the members need to work together closely. Lastly it claims the most effective and efficient method of conveying information to and within the development team is face-to-face conversation (*AM₆*). So it recommends an explicit way to achieve close collaboration, emphasizing its importance.

Scrum explicitly advocates collaborative teams of developers [132]. Next to this it defines a pre-game phase in which the project planning and process tracking is made a collaborative activity. Extreme programming also focuses on the importance of teams that collaborates closely. The first value of XP is that good communication helps reduce misunderstandings and increase cooperation in teams. Beck states: *"Problems with projects can invariably be traced back to somebody not talking to somebody else about something important"* [12]. The final value of XP which emphasizes the importance of respect within the team and that the team gets along well, is quite related to the importance of close collaboration as well. This high priority of close collaboration is also reflected in a couple of practices of XP. For instance, advocating the whole team works in a single room (*XP₁*) is a way to try and coerce the teams into collaborating closely. Picking teams in which the members supplement each other (*XP₂*), guaranteeing the continuity of effective teams (*XP₁₆*), and using an informative workspace (*XP₃*) and user stories (*XP₆*) to make knowledge sharing easier, are ways to try and achieve close collaboration as well. Finally, one of the best known practices to achieve close collaboration in XP is pair programming (*XP₅*), in which two team members actually work together on a single task.

Relation to distance

Having development teams that collaborate closely could be a way to deal with distance because the negative affects of being geographically, temporally or social-culturally separated are of less concern to a close development team.

A₂ Short iterations, frequent builds and continuous integration**Short description**

This aspect concerns that development is done by delivering incremental components of business functionality in so-called iterations. Developing a product in this fashion helps

to keep focus on short term goals and to create working software quickly. During these iterations the work should be integrated and build as frequent as possible to be able to detect and resolve problems early, when they still can be dealt with, with relative ease. After each iteration ends and before the next one begins, the process that is used should be adapted to work even better in the next iteration [56].

Relation to the agile methodologies

The processes specified in each of the agile methodologies discussed in this chapter concern a development process consisting of short iterations with frequent builds. While the Agile Manifesto does not explicitly specify a process, it does state that the highest priority is to satisfy the customer through early and continuous delivery of valuable software (AM_1), that software needs to be delivered frequently, with a preference to a short time scale (AM_3) and that working software is the primary measure of progress (AM_7).

XP also has such practices: It dictates both a weekly (XP_7) and quarterly (XP_8) build, the fact that iterations should also consist of work which can be omitted if deadlines cannot be reached (XP_9) and that the design of the system should be done incrementally (XP_{13}). Next to this it offers many practices related with continuous integration (XP_{12}). These practices are: being able to build the system in ten minutes (XP_{10}), use test-first programming to be able to always test the current system (XP_{11}), doing root-cause analysis when a defect is detected (XP_{18}) and using daily (XP_{22}) and incremental (XP_{15}) deployment to avoid the problems and risks associated with big deployments. Continuous integration is closely related to feedback, as by continuously integrating and building, feedback can be acquired more often. This regards both feedback acquired from colleagues and the customer as well as feedback acquired from the results of tests. Acquiring timely feedback is one of the core values of eXtreme programming. Feedback is reflected in the Agile Manifesto as well, as can be gathered from principle AM_9 , which subscribes the continuous attention to technical excellence and good design.

The final part of this aspect left to relate to existing agile methodologies concerns the self-adaptivity of the process. The Agile Manifesto states that the team should reflect, at regular intervals, on how to become more effective and then tune and adjust its behavior accordingly (AM_{12}). In Scrum this is done in what is called a *Sprint Retrospective meeting* which is held at the end of every sprint. In ASD and Crystal methods, self-adaptivity is most noticeable [56]. In XP, it seems to be disallowed by the rather rigid rules, but this is only a surface impression since XP does encourage people to tune the process. Review of the process however, is not an explicit part of the process in eXtreme programming, although there are suggestions to change this [56].

Relation to distance

Using short iterations with frequent builds and continuous integration is also an aspect of agile software development which could help deal with the distance in a globally distributed

setting. The increased amount of feedback, for instance, could decrease the socio-cultural distance because of the motivating factor this offers. Also the things that are more difficult because of the global and temporal displacement, insight in the progress of the project and managing the versioning of the system, can be alleviated by this aspect.

A₃ Decentralizing the decision making

Short description

In agile software development part of the decision making is moved to the developers. Management is still needed to remove roadblocks standing in the way of progress. The development team, however, is entitled to make certain technical decisions without involvement of the management. This aspect also concerns the decisions made by a subgroup of a development team rather than an individual.

Relation to the agile methodologies

This aspect is less directly found in the methodologies discussed. Most agile methodologies however, advise having faith in the abilities of the developers. The Agile Manifesto for instance states that projects should be built around motivated individuals that receive the environment and support they need and that they should be trusted to get the job done (AM_5). It also emphasizes the power of self-organizing teams (AM_{11}) and that such teams need to be trusted to get the job done. This is exactly what this aspect seems to be about most; management should recognize the expertise of the developers [56]. In XP this aspect does not correspond to a specific practice but in XP the developers must be able to make all technical decisions [56] and this is quite what this aspect is about. Next to the developers being trusted by management to make decision they should also be courageous enough to do so. This is reflected in the courage value of eXtreme programming: *"The development team needs to have courage in its actions and decision making to perform the right actions and make the right decisions"* [141].

Relation to distance

This aspect also can help to deal with the distances associated with GSD projects. For one, being trusted and receiving autonomy motivates developers and thus decreases the social-cultural distance. Next to this, problems associated with the geographical and temporal distance can be avoided by taking away the bureaucratic communication overhead.

A₄ Customer involvement

Short description

This aspect involves the active participation of the customer in the development of the system. This involvement is used to acquire feedback on the current implementation of the system and to further clarify the requirements of the system to be built. It is important to note that the customer has the right to change requirements during the entire project.

Relation to the agile methodologies

The Agile Manifesto emphasizes the importance of the customer in the development process. Again we refer to the fact that the Agile Manifesto regards satisfying the customer through early and continuous delivery of valuable software as having the highest priority (AM_1). Also it states that changing requirements should be welcomed, even late in development (AM_2). Besides this also in the values of the Agile Manifesto the importance of this aspect can be seen as these both mention the importance of customer collaboration and responding to change.

In eXtreme programming there is a practice which states that the customer should be available to clarify the requirements, is a subject matter expert, is empowered to make decisions about the requirements and their priority and writes the acceptance tests (XP_{14}). Another way XP aims to keep the involvement of the customer high, is the use of stories (XP_6). These are short statements regarding the desired functionality of the product, visible to the customer, which the customer prioritizes. Furthermore eXtreme programming states the contract which is agreed with the customer should allow for on-going negotiation regarding the scope of the project to allow for further flexibility (XP_{23}). Finally, XP also recommends charging on a pay-per-use basis as to obtain feedback from the users based on their usage patterns (XP_{24}). With respect to customer involvement in Scrum Schwaber states the following: *"The SCRUM approach, however, welcomes and facilitates their controlled involvement at set intervals, as this increases the probability that release content and timing will be appropriate, useful, and marketable"* [132].

Relation to distance

This aspect can also be beneficial with respect to helping to cope with the distances faced when developing globally distributed. This is because getting the appropriate feedback motivates the development team and motivated individuals feel more like a team, thus decreasing the social cultural distance.

A₅ Collective ownership of work

Short description

This aspect concerns that what is produced is the result of the entire team, and not an individual. No single team member owns, or is responsible for a specific code segment and all work can be changed by the entire team, without explicit permission. This aspect, however, is not restricted to all team members being able to change part of the work without explicit permission. It also concerns the development team having a shared vision and responsibility of the system to be built. The whole team is creating something together, aiming for a single collaborative goal and is collectively responsible that this goal is reached.

Relation to the agile methodologies

This aspect is again found in a less direct manner in both the Agile Manifesto and Scrum. Chao et al. [33] states: "... several agile methods (e.g. XP and Scrum) imply that explicit knowledge including designs and models should be collectively owned". It is true that this aspect is merely implied in the Agile Manifesto. The Agile Manifesto states that the best architectures, requirements and design emerge from self-organizing teams (AM_{11}) and these kind of teams are teams with a shared responsibility with the respect to the work they are performing. The principle which states projects should be built around motivated individuals who should be given the environment and support they need and should be trusted to get the job done (AM_5) implies the project is the shared responsibility of the entire development team.

In eXtreme programming there is a practice which explicitly states that all code is shared by the development team and thus can be altered by any team member (XP_{19}). It also specifies a practice that a single code base should be used (XP_{21}) which makes it easier for all team members to have constant access to all work. Besides this, eXtreme programming dictates a cross functional team (XP_2). This also supports this practice because a team must be knowledgeable in all fields required in the project to be able to share the responsibility of the project.

At first glance it seems Scrum is incompatible with collective ownership because it defines the role of product owner. However, Judy et. al states: "*a capable Scrum Product Owner and performing team can build a spirit of collective ownership over all aspects of a product lifecycle while still fulfilling the Scrum roles. This collective product ownership is close to the lean product development origins of agile which values the insights into products held by those who build them and positions an organization for sustained excellence and product innovation.*" [86].

Relation to distance

By incorporating this aspect into the development process the socio-cultural distance could be decreased since teams that work together towards a common goal are often more cohesive.

A₆ The most important artifact is the system to be built

Short description

This aspect concerns that working software is the most important goal of the project and other matters, like documentation, are inferior to it. Spending a lot of time on updating non-essential artifacts should be prevented.

Relation to the agile methodologies

One of the values of the Agile Manifesto states "*working software over comprehensive documentation*" which is exactly what this aspect is about. This aspect is also reflected in two of its practices. Firstly it states working software is the primary measure of progress (AM_7) and secondly it states the highest priority is to satisfy the customer through early and continuous delivery of valuable software (AM_1). Extreme programming also shows the higher importance of software by stating only the code and tests are artifacts that should be maintained (XP_{20}).

Relation to distance

This aspect does not offer specific benefits when working globally distributed which it does not offer when working co-located.

A₇ Favoring simplicity

Short description

Agile software development favors simplicity, or the maximization of the amount of work not done [16]. This aspect concerns looking for the simplest working solution first and improving it later only if the need arises. This aspect is based on the assumption that requirements and other matters with respect to the project are likely to be changed in the future so that it is rarely worth to implement things supposed to be helpful at some point in the future.

Relation to the agile methodologies

The relation of this aspect to the methodologies is rather direct. The Agile Manifesto states that it is essential (AM_{10}), and eXtreme programming states simplicity as one of its core values. It does not mention simplicity directly in one of its practices, however.

Relation to distance

This aspect does not offer specific benefits when working globally distributed which it does not offer when working co-located.

A₈ Sustainable pace of development

Short description

This aspect is concerned with the fact that people should not work excessive overtime for long periods of time. The motivation behind this is that un-energized people produce less and lower quality work than energized people. DeMarco states: "*Extended overtime is a productivity reducing technique*" [48]. The idea is that the development speed is such that all people involved in the project are able to maintain a constant pace indefinitely.

Relation to the agile methodologies

Again the relation to both the Agile Manifesto and eXtreme programming is rather direct. The Agile Manifesto prescribes that the sponsors, developers and users should be able to maintain a constant pace indefinitely (AM_8) while XP dictates that the members of the development team must stay energized (XP_4) and thus not work excessive overtime for long periods of time. Finally, eXtreme programming also mentions that a team which grow in capacity, due to experience, should keep a constant workload but should shrink in size to keep the workload of the individual members of the team constant (XP_{17}).

Relation to distance

This aspect does not offer specific benefits when working globally distributed which it does not offer when working co-located.

Overview

In table 6.1 an overview is given of the relation between the principles of the Agile Manifesto and the aspects of agile software development we defined in this section. In table 6.2 an overview is given of the relation between the practices of eXtreme programming and these same aspects.

6. AGILE METHODOLOGIES

Practice	A1	A2	A3	A4	A5	A6	A7	A8
AM1		X		X		X		
AM2				X				
AM3		X						
AM4	X							
AM5			X		X			
AM6	X							
AM7		X				X		
AM8								X
AM9		X						
AM10							X	
AM11	X		X		X			
AM12		X						

AM1	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
AM2	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
AM3	Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
AM4	Business people and developers must work together daily throughout the project.
AM5	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
AM6	The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
AM7	Working software is the primary measure of progress.
AM8	Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
AM9	Continuous attention to technical excellence and good design enhances agility.
AM10	Simplicity—the art of maximizing the amount of work not done—is essential.
AM11	The best architectures, requirements, and designs emerge from self-organizing teams.
AM12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work
A6	The most important artifact is the system to be built
A7	Favoring simplicity
A8	Sustainable pace of development

Table 6.1: Relation between the Agile Manifesto and the aspects of agile software development

Practice	A1	A2	A3	A4	A5	A6	A7	A8
XP1	X							
XP2	X				X			
XP3	X							
XP4								X
XP5	X							
XP6	X			X				
XP7		X						
XP8		X						
XP9		X						
XP10		X						
XP11		X						
XP12		X						
XP13		X						
XP14				X				
XP15		X						
XP16	X							
XP17								X
XP18		X						
XP19					X			
XP20						X		
XP21					X			
XP22		X						
XP23				X				
XP24				X				

XP1	Sit together	XP13	Incremental design
XP2	Whole team	XP14	Real customer involvement
XP3	Informative workspace	XP15	Incremental deployment
XP4	Energized work	XP16	Team continuity
XP5	Pair programming	XP17	Shrinking team
XP6	Stories	XP18	Root cause analysis
XP7	Weekly cycle	XP19	Shared code
XP8	Quarterly cycle	XP20	Code and tests
XP9	Slack	XP21	Single Code Base
XP10	Ten-minute build	XP22	Daily deployment
XP11	Test-first programming	XP23	Negotiated scope contract
XP12	Continuous integration	XP24	Pay-per-use

A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work
A6	The most important artifact is the system to be built
A7	Favoring simplicity
A8	Sustainable pace of development

Table 6.2: Relation between eXtreme programming and the aspects of agile software development

Chapter 7

Global software development combined with aspects from agile methodologies

When carrying out software development projects professionally, a process is used to prevent the project from lapsing into chaos. What kind of process should be used will depend on the specific requirements and constraints of the project. Barry Boehm argues: *"Both agile and plan-driven methods have a home ground of project characteristics in which each clearly works best, and where the other will have difficulties. Hybrid approaches that combine both methods are feasible and necessary for projects that combine a mix of agile and plan-driven home ground characteristics."* [23]. For globally distributed projects, Grinter et al. [64] suggest using a structured, plan driven process, as a way to coordinate such projects. There are however two reasons why less plan-driven, more agile practices could be useful for GSD projects.

1. The agile methodologies could be beneficial to global software development in the same way they are useful to software development in general. This regards that in general, they are able to better cope with uncertainty and changing requirements in projects than plan-driven approaches. Examples of such projects are projects developing genuinely novel products since these projects are often faced with uncertainty regarding, both requirements and implementation technologies, and subcontractors or partners need to be involved long before these uncertainties can be resolved. This reason alone would justify attempting to facilitate the use of agile methods in a distributed setting.
2. The application of practices from agile methodologies could result in a reduction of the negative influence of distance on communication, coordination, and control in a GSD context. This is plausible since, on the one hand, the most important challenges of GSD lie in the complexity of maintaining good communication, coordination and control when teams are dispersed [4], while on the other hand, agile methodologies emphasize communication and as a result, reduce coordination and control overhead [82, 109].

There is, however, another side to this story, as frequent collaboration and face-to-face communication belong to the basic principles of agile methods [109]. The literature on XP, for instance, emphasizes that it is important to have the team members physically located close to each other [13]. So when attempting to incorporate agile practices into GSD this will lead to various challenges the original agile methodologies did not have to deal with, due to assuming the various team members would be physically close to each other [91, 109].

We refrain from elaborating any further on the first reason why agile methodologies could be beneficial to GSD because this is not specific to GSD. We direct the interested reader at [23, 21] for a more thorough explanation. In the rest of this chapter we will discuss the impact incorporating aspects of agile development approaches, introduced in the previous chapter, have on the perceived distances and the consequences of these distances, in a distributed setting. We will do this by first discussing how incorporation of each of the aspects that influence distance can reduce the distances faced. Then we will discuss the main challenges faced when using agile software development practices in a distributed environment and how these challenges can be faced. In the last section we will conclude by summarizing the benefits, challenges and potential solutions discussed in this chapter in three overview tables for easy reference.

7.1 How aspects of agile methods can reduce distance in a GSD context

As discussed in chapter 2, all the challenges associated with global software development originate from the existence of three kinds of distances:

- *Geographical distance*
The dislocation in space experienced by two actors wishing to interact.
- *Temporal distance*
The dislocation in time experienced by two actors wishing to interact.
- *Socio-cultural distance*
The dislocation in understanding of the values and normative practices of other actors.

The combination of these distances is what makes global software development such a complex task [82]. A way to deal with the challenges commonly faced in GSD would be to reduce these distances themselves, reduce the consequences of the existence of these distances or help to cope with the consequences of these distances. This section will discuss the influence the *aspects* defined in chapter 6 have on the three types of distances.

Aspect 1: Close collaboration among the members of the development team

Temporal

Close collaboration leads to good report between colleagues. Good report between colleagues means they are more willing to compromise with respect to working times. An

example of this effect with respect to XP pair programming is mentioned by Agerfalk et al.: *"... people were flexible and, even though there was a delay in response, individual developers tried hard to spend as much time as possible with the distributed pair programmer"* [82]. Hence the negative influence of the *temporal* distance is reduced since the time overlap is increased in comparison to both team members just working normal office hours and not being flexible. Next to this, colleagues will also be more inclined to help out a colleague with a small task, like answering a quick question, when at home. This helps to reduce the challenge of *delayed communication*.

Geographic

The main challenges which are a direct consequence of the geographic distance in GSD are the lack of informal communication and the increased effort to initiate contact. At the same time close collaboration between team members generally increases informal communication and eases initiating contact. Therefore we can say close collaboration between team members *reduces* the main consequences of the geographical distance.

Socio-cultural

Close collaboration, does not just lead to good report between colleagues, it also leads to a better mutual understanding between them. Therefore close collaboration actually reduces the socio-cultural distance between team members. Agerfalk et al. again mentions XP pair programming as an example: *"... the practice of pair programming improved individual commitment for enhancing mutual understanding between team members"* [82]. Because of this, challenges mainly caused by the socio-cultural distance, like *"lack of team cohesion"* and *"lack of trust"*, are being dealt with by having a closely collaborating team.

Aspect 2: Short iterations, frequent builds and continuous integration

Socio-cultural

The main distance this aspect of agile development targets is the socio-cultural distance. This is because short iterations, frequent build and continuous integration leads to feedback. This feedback motivates the developer [109] and motivated developers feel more like a team. Next to this, seeing high quality work early and frequently results in trust [109]. Both feeling more like a team and increased trust reduce the socio-cultural distance caused by working globally distributed.

Temporal and geographical

In GSD projects in general, it is quite hard to have a good idea of the progress of work, mainly due the temporal and geographical distances which make direct communication difficult. Because of this, problems can be detected too late, and this can result in time and resources being wasted. Continuous integration can improve upon this situation because it leads to transparency of the progress [135, 109]. Another issue in GSD projects aggravated by geographic and temporal distance is configuration management: the management of the

evolution of the complete system [138]. Continuously integrating, in contrast to integrating everything at the end of the project, makes configuration management less of an issue [135]. This is because most problems faced when continuously integrating the system are faced in isolation and isolated problems are easier to identify because they cannot interact with each other [58].

Aspect 3: Decentralizing the decision making

Temporal and geographical

Because of this aspect, both temporal and geographical distance becomes less of an issue because, developers can take certain decisions without having to confer with management which could be located in another part of the world. Next to this, the geographical and temporal distances often make it hard to make proper estimations regarding the progress of the project. By letting the people performing the work make these estimations they are often more accurate [135].

Socio-cultural

Giving the individual developers and development teams autonomy is a great motivator and allows people to be more productive [57]. As mentioned earlier, this increase in motivation leads to developers feeling more like a team, which reduces the socio-cultural distance between them. Next to this, allowing teams and individual developers to make their own decisions conveys trust. According to Zand [144], one of the most powerful ways to show your own trustworthiness is to trust the other.

Aspect 4: Customer involvement

Socio-cultural

In the description of aspect two, we described that constant feedback decreases the socio-cultural distance between team members, because feedback motivates the developers and seeing high quality work early and frequently results in trust. Feedback, however, is much more valuable when it is given by the actor for which the system is being built: the customer. Only the customer can decide if a feature is implemented in the way he had in mind. In agile software development high customer involvement is a general practice. Agile methods generally advocate that the customer becomes part of the development team during part of, or even the entire, development process.

Aspect 5: Collective ownership of work

Socio-cultural

In chapter 4 the characteristic that a real team should possess according to Carmel [29] are mentioned. Two of those characteristics are:

- *Collective responsibility for its products*

- *Shared responsibility for managing its work*

Hence, collective ownership of work will lead to closer development teams, which, in turn, reduces the socio-cultural distance between its members.

7.2 Problems with incorporating agile aspects into GSD

The need for frequent and rich informal communication is larger when developing in an agile fashion than when following more plan-driven methodologies. This is however also more difficult when developing globally distributed due to the existence of the geographical, temporal and socio-cultural distances. Often agile methods even advocate the need for physical proximity, XP for example, emphasizes that it is important to have the team members physically located close to each other [13, 91]. This leads to challenges faced in non-agile distributed development to worsen and the emergence of new challenges when attempting to use an agile methodology in a distributed setting. In the previous section we discussed the benefits an agile methodology could offer in a distributed setting. We discussed how a number of aspects of agile methodologies we identified as being able to influence distance in chapter 6, could offer advantages with respect to dealing with the three distances faced in distributed development. In this section we will return to these aspects but this time we will discuss how working globally distributed makes it harder to achieve these aspects and initial ideas on how to overcome these difficulties.

Aspect 1: Close collaboration among the members of the development team

Geographical and temporal

The geographical and temporal distances between developers in GSD projects causes frequent communication between developers to be difficult [135, 57, 125, 109]. Because of this, close collaboration is also harder to accomplish than when the development team is collocated since frequent communication is essential for close collaboration. Basically, there are two ways to deal with this issue, which must be used in conjunction in order to achieve the best possible result:

1. *Improve the communication itself*
2. *Facilitate other means for knowledge sharing*

Examples of ways to improve the actual communication between developers are:

- *Synchronize work hours*
By synchronizing the work-hours of the development team as much as possible the temporal distance can be minimized [125]. This will make a larger amount of synchronous communication between developers possible. The work-hours should at least be synchronized to the degree that regular short status meetings can be held with the entire development team [57, 125].

7. GLOBAL SOFTWARE DEVELOPMENT COMBINED WITH ASPECTS FROM AGILE METHODOLOGIES

- *Provide for informal communication through formal channels*

By formalizing the informal communication in a way that it utilizes formal channels the reliance on informal communication is reduced [125]. An example of doing this is by designating someone as the point of contact for each team. This person is responsible for facilitating communication across the teams and is to work closely with both development and project management teams on a daily basis [125, 94, 92]. This solution will also give the opportunity to incorporate a little more control into the development process when it is extended with a more formal structure for reporting and responsibility [125]. When doing this, potential problems in the collaboration between the various development sites are more easily visible and can be dealt with.

- *Maintain constant communication*

Even though constant face-to-face communication is infeasible when developing geographically distributed, constant communication is still feasible by making use of other means of communication [125]. When selecting a means of communication synchronous means, like telephone calls and instant messaging, should be favored over asynchronous means, like SMS and email. However when it is impossible to use synchronous communication asynchronous communication is a sufficient surrogate as long as the communication loops are short [94].

Examples of facilitating for knowledge sharing are:

- *Make use of universally-accessible knowledge base*

By making use of a universally-accessible knowledge base like Wikis, message boards and repositories the progress of the various teams is more transparent [135, 27, 125, 94]. This will help decrease misunderstandings between development teams and makes sure knowledge is consistent between all development teams. The amount of information that is documented in such a knowledge base should be larger than when developing in a non-distributed setting, yet not as excessive as with plan driven methodologies. The idea is to document to a level of detail which is sufficient at the current state of the project [135, 57, 125].

- *Perform extensive requirements-communication*

It is important for all developers in a project to have a view on the requirements which is as similar as possible [116]. Because of this, when developing dispersed across the world, more ceremony needs to be put into communicating requirements [57]. Fowler [57] gives the example of using acceptance tests as ways of communicating requirements. When doing this one development team would describe the features which needed to be implemented in the next iteration. Another team would then use these descriptions to write test scripts, which can be either manual or automated. As the scripts are developed, both the development teams coordinate by means like email and instant messaging as well as regular conference calls to review if there is consensus about the requirements.

Socio cultural

In distributed development, participants at different sites are less likely to perceive themselves as part of the same team than with collocated development. Because of this, team members find it hard to have faith in the good intentions of remote colleagues [91] and there can be a lack of team cohesion [125]. These things make close collaboration between developers at different geographical locations more difficult. In order to deal with this, trust and shared understanding must be build between team members. Examples of way to achieve this are:

- *Select teams that work well together*
By selecting teams of people that have had prior working relationships with each other the problems described above are greatly alleviated [125]. If this is not possible, selecting teams in such a way that the people that need work most closely together have a good working relationship would still make close collaboration between the teams easier in general.
- *The periodic exchange of team members between sites*
By periodically rotating team members between sites, more developers get to work together on a single location. Because of this, the sense of being a single community among developers, is increased, as well as the trust in remote colleagues [57, 117, 27, 125]. When following this policy, basically there are three variables to consider:
 - *The amount of people to exchange*
 - *Which people to exchange*
 - *The amount of time these people will stay at another site*

Fowler [57] suggests the concept of *ambassadors*. In this concept at least one member of each team is present at another site at all times to represent his team. Fowler suggests periods of exchange of several months and choosing people for the role of ambassador that do not mind being away from home for several months. He also suggests that is particularly important for project managers to spent some time as *ambassadors* because of the very nature of their function. In order to be able to help resolve conflicts and flush out problems before they become serious, experience with working in the different teams is really important [57]. Layman et al. [94] suggests that having a having a key member of one team physically located with the other team can provide an essential two-way communication conduit. They argue, because this person will communicate often with both teams because of his importance in the process, he will increase the connection between these teams. Braithwaite et al. [27] and Poole [117] both suggest shorter, but more frequent exchanges of different people in order for more members of the teams to have physically met each other and built a working relationship.

Aspect 2: Short iterations, frequent builds and continuous integration

Performing short iterations, frequent builds and continuous integration is harder in a distributed than in a collocated setting for a couple of reasons. For one, the lengths of the iterations must be increased to compensate for the time lost due to communication overhead [57]. Both Fowler [57] and Ramesh et al. [125] suggest adopting an iteration length of at least two weeks. Fowler however, also reports that the iteration length should stay below two months.

Secondly, iterations using an agile development approach often define stand up meetings at the start of the iteration and regular short status meetings during the iterations, in which everyone, or a large portion of the team members, should take place. Carrying these practices over to the distributed environment is important for proper coordination between teams [57] but also difficult, both due to the different work-hours of different teams [57] and because it is often impossible to perform such meetings in a single room due to the geographic distance between the teams. Because of this, these meetings have to be tailored to work when working distributed [57]. We will discuss these adaptations based on the distance which makes it necessary:

- *Geographic*

The geographic dislocation is best dealt with by providing team members with as many communication media as possible in order for the communication between them to be as rich as possible [27]. An example of this is providing a remote colleague a link to example code during a video conference [27].

- *Temporal*

With respect to the difference in working hours the teams should compromise so all teams share an equal amount of discomfort [57]. Also, because the time these meetings can take is limited, when a remote team member has to go through something before the discussion can continue, it is best to leave the point for now and agree to return to the issue at a later time when the remote team member has had time to do this outside of the meeting [27].

Lastly, configuration and version management is also more difficult in a distributed setting [109]. Even though teams at different geographical locations are working on the project, possibly at the same time, an unambiguous global view of the current system needs to be maintained. To accomplish this, a common code base should be used [117, 109]. This would need to consist of both a source control system and common build environment which enables all developers to build and test their code [117]. This however, also has some extra difficulties associated with it when working in a distributed setting. These difficulties originate from the geographic and temporal distances and we will discuss these separately:

- *Geographic*

Because of the use of a single code base and multiple, geographically distributed sites, some sites could experience suboptimal performance of the common code base. This

could both be caused by communication lines which are too narrow, but, in certain parts of the world, they might also periodically fail altogether [57, 109]. There are three basic ways to deal with this problem:

1. *Locate the code-base server on the site which has the highest amount of developers working on the project [57].*
2. *Locate the code-base server on neither of the sites, but rather place is strategically in the middle, so no one suffers from extreme performance issues.*
3. *Make use of a clustered code repository, with each site having access to a code-base server. Of course the clustered code repository must be such that the global integrity is maintained [57].*

- *Temporal*

If the developers have different work-hours, if some developer breaks the build at the end of his work-day and goes home this causes the team that works after having to deal with the broken build without having access to the developer that caused it. In the worst case this might cause development of an entire site to stall for a day. The way to deal with proposed in the literature, is to arrange for developers that committed changes to the mainline, do not go home until it is confirmed that their changes resulted in a successful build [135, 57]. Another problem caused by teams working at different times of the day, is that this makes it tougher to find a time to take the server down for updates or backups.

Aspect 3: Decentralizing the decision making

Socio cultural

In order to use decentralization of decision making properly within a project, the developers should be autonomous and independent enough to actually make decisions on their own. Particularly in Asian cultures, but also in others, this is often a problem because they are often trained to listen to their superiors and not make decisions on their own [57]. This is quite a problem, as parting people from a life-long believe is often a process which takes quite some time. Fowler [57] suggests however, that in the end people with cultural heritage of this kind will come around and actually relish the freedom they have received. Meanwhile, it is best for superiors to keep in mind the cultural heritage of these people and be prepared to deal with any problems that this might cause.

Geographic and Temporal

The other side of this story is that it can be hard to make decisions decentralized when globally and temporally distributed from the rest of the development team. This is because the developer which has to make the decision can reach better decisions when he has access to certain types of information, depending on the decision that needs to be taken, and this information can be harder to acquire when the development team is distributed geographically and temporally.

Aspect 4: Customer involvement

Geographic and Temporal

In distributed development, the customers may be located far away making frequent interaction difficult to arrange. Having the customer on site is even harder in this situation and if it can be arranged the on-site customer might gradually lose connection to his or her own environment [109]. A way to deal with this is by arranging frequent regular meetings with the customer. For these meetings the customer should travel to the development site as often as possible and otherwise make use of as many communication media as possible in order for the communication between them to be as rich as possible. Kircher [91] mentions the use of video conferences, arranged via email, in distributed XP effectively involves the customer throughout the project. Another example is having regular integrated builds accessible to the customer. The customer is then able to test out the most recent version of the system and submit feedback. This might not be quite as immediate as co-location, it still allows the customer to correct any misunderstandings quickly; as well as allowing them to refine their own understanding of the requirements [57]. Still, communication via such media is only a substitute for face-to-face communication and if the customer is located in a different part of the world time differences also have to be dealt with. In order to deal with these issues it is possible to make use of a *proxy-customer* which is not actually the customer but rather someone who plays the role of on-site customer [135, 94, 92]. When selecting proxy customers it is best to select people with a combination of interests and abilities in technical and business spheres. The proxy-customer must be able to interact equally well with the technical and the business project members but does not necessarily have to be experienced [135]. He or she should be able to make conclusive decisions on project functionality and scope, and have a vested interest in the project [94, 92].

Aspect 5: Collective ownership of work

The main problem with incorporating this aspect of agile methodologies in a GSD context is that the very idea might conflict with certain ideologies and cultural beliefs. Berteig [18], for instance, discusses teaching a course on agile development in Romania. Several of the Romanians following the course felt that the collective ownership of work closely resembled elements from the communistic ideology, to which they objected. Another problem with incorporating this aspect in a distributed setting is that all work should be accessible by everyone from anywhere in order for all the work to truly be collectively owned. The solution to this is very similar to the common code base discussed earlier in this section so we will refrain from describing this here any further.

7.3 Overview

In this section we will provide an overview of the benefits and challenges discussed in the rest of this chapter, as well as the proposed solutions to these challenges, for quick reference. Table 7.1 provides an overview of *the benefits of incorporating agile aspects into GSD work*, table 7.2 of *the challenges faced when incorporating agile aspects into GSD*

work and finally table 7.3 sums up *the proposed solutions to the challenges faced when incorporating agile aspects into GSD work*.

Aspect	Distance	Description
A1	Temporal	B1: More overlap in working time
	Geographical	B2: Increase in informal communication
	Socio-cultural	B3: Better mutual understanding
A2	Socio-cultural	B4: Feedback motivates the developer
		B5: Seeing high quality work early and frequently results in trust
	Geographical and Temporal	B6: The process is more transparent
		B7: Configuration management is less of an issue
A3	Geographical and Temporal	B8: Less control needed
A4	Socio-cultural	B9: Better progress estimations
		B10: Autonomy motivates the developers
		B11: Being trusted results in trust
A5	Socio-cultural	B12: The customer can give valuable feedback
	Socio-cultural	B13: Increase in team cohesion

A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work

Table 7.1: Benefits of incorporating agile aspects into GSD work

7. GLOBAL SOFTWARE DEVELOPMENT COMBINED WITH ASPECTS FROM AGILE METHODOLOGIES

<i>Aspect</i>	<i>Distance</i>	<i>Description</i>
A1	Geographical and Temporal	C1: Frequent communication is difficult
	Socio-cultural	C2: Team members are less likely to perceive themselves as part of the team
A2	All	C3: Less can be achieved in a certain amount of time due to communication overhead
	Geographical and Temporal	C4: Stand-up and status meetings are more difficult
		C5: Unambiguous global view of the current system is more difficult to be maintained
A3	Socio-cultural	C6: Some developers might not be autonomous and independent enough to make decisions themselves
	Geographical and Temporal	C7: Having access to sufficient information in order to reach correct decisions is more difficult
A4	Geographical and Temporal	C8: Frequent interaction with the customer is difficult
		C9: When the customer is placed on-site, he might gradually lose connection with his own environment
A5	Socio-cultural	C10: People might object to the notion
	Geographical and Temporal	C11: It is more difficult for all work to be accessible to everyone at any time from any location
A4	Socio-cultural	B12: The customer can give valuable feedback
A5	Socio-cultural	B13: Increase in team cohesion

A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work

Table 7.2: Challenges faced when incorporating agile aspects into GSD work

Challenge	Proposed Solution
C1	S1: Synchronize work-hours
	S2: Provide for informal communication through formal channels
	S3: Maintain constant communication
	S4: Make use of universally-accessible knowledge base
	S5: Perform extensive requirements-communication
C2	S6: Select teams that work well together
	S7: Exchange team members between sites periodically
C3	S8: Use longer iterations
C4	S9: Make the communication as rich as possible by using multiple communication media
	S10: Have the team members compromise with respect to the time these meetings are held
	S11: Only discuss matters that can be discussed immediately
C5	S12: Locate the code-base server in a way as to avoid difficulties [see previous section]
	S13: A developer that commits changes should not go home until it is confirmed he or she did not break the build
C7	Similar to Challenge 5
C8	S14: Arrange for frequent regular meetings with the customer
	S15: Make builds available to the customer on a regular basis
	S16: Assign a proxy-customer
C11	Similar to Challenge 5

C1	Frequent communication is difficult
C2	Team members are less likely to perceive themselves as part of the team
C3	Less can be achieved in a certain amount of time due to communication overhead
C4	Stand-up and status meetings are more difficult
C5	Unambiguous global view of the current system is more difficult to be maintained
C7	Having access to sufficient information in order to reach correct decisions is more difficult
C8	Frequent interaction with the customer is difficult
C11	It is more difficult for all work to be accessible to everyone at any time from any location

Table 7.3: Proposed solutions to the challenges faced when incorporating agile aspects into GSD work

Part IV

Supporting Agile Global Software Development with technology

Chapter 8

Types of technology which support GSD

In order to properly discuss how to support agile global software development with technology, different types of technological support, beneficial in that specific environment, should be defined. We will define these categories based on what requirements are needed in supporting technology for agile GSD. To derive these 'so-called' *requirement categories* for agile GSD we will first propose such categories for GSD in general in section 8.1. Following this we will discuss which benefits and challenges, associated with GSD in general, can be supported by which of the requirement categories we defined. By doing this we verify that each of the requirement categories supports at least one of the challenges and benefits. In the second section we will propose a similar categorization. This categorization will concern the types of technological support applicable to assist in dealing with problems and exploiting possibilities in incorporating agile aspects into the GSD development process. To do this we will discuss to what extent the requirement categories of GSD in general are applicable in this context and how they should be altered or extended to be fully applicable.

8.1 Types of technology which support GSD in general

In this section a categorization of technological aid to help with GSD is proposed. The applicability of the proposed requirement categories is argued by showing that each of these categories support at least a single characteristic of GSD. To do this, first we discuss which benefits and challenges of GSD, defined in chapter 3 and 4, can be supported by technology. Subsequently we discuss technological support from which requirement categories can be used to support each of these challenges and benefits.

The first process that collaboration technologies should support is the communication between team members. This is required because the software development process does not operate on mandate, orders and edicts but on the self-managing capabilities of the developers [29, 103]. Hence the technology needs to provide an infrastructure for collaborative sessions, making it easier to contact colleagues, in order to support this self-managing char-

acter of software development [29, 103]. Another problem in distributed software development is the risk of *re-inventing the wheel*. This risk can be reduced when tools are used to create a rich and up-to-date *project memory* including versioned files, change histories and technical documentation [29, 103, 70]. Thirdly, in a globally distributed project it is much harder to understand what other project members are doing than in a co-located project [29, 103, 70]. Therefore project transparency is necessary in order to coordinate effectively with the members of the team. Technology which provides status information about tasks, people and other dynamic team information to all team members at different sites can increase the transparency of the project. The final process we discuss is quality assurance. Quality assurance is essential to monitor and guarantee the quality of the product. Technology which supports quality assurance functions, such as bug tracking and requirements tracking can have a positive influence on the total quality of the product [29]. From this enumeration the following requirement categories can be derived:

R₁ Facilitate direct contact between colleagues

Technological support which facilitates direct communication between two or more actors

R₂ Facilitate knowledge sharing among colleagues

Technological support which facilitates the sharing of *technical* project knowledge

R₃ Facilitate transparency of the project status

Technological support which facilitates the sharing of *organizational* project knowledge

R₄ Facilitate quality assurance

Technological support which facilitates quality assurance functions to monitor and guarantee the quality of the product

As announced we will start by enumerating all the benefits of GSD and by examining the possibilities for technological support for each of the benefits, this is shown in table 8.1. We can see that there are four benefits which can be exploited by means of technology; these benefits have a single plus in the technological support (TS) column. Take for example the benefit of being able to use specialized or skilled people; this benefit could be further exploited by technology which makes the information of the specialists more easily accessible to anyone who is interested. Another benefit which can be further exploited by means of technological support is the time to market reduction. Technology adapted to *round-the-clock* development can further help to decrease the time to market than common development tools. For all benefits listed in table 8.1 non technological support (NTS), e.g. changes in approaches, can exploit the benefits of GSD.

Following the categorization of the benefits into non technological support and technological support the same categorization is applied to the challenges of GSD, this shown in table 8.2. Most of the cultural challenges of GSD can only be alleviated by non technological support, the only cultural challenge that can be alleviated by technology is the challenge

Benefit	NTS	TS
Handle the increased product complexity	+	+
Usage of specialized or skilled people	+	+
Access to a sufficiently large workforce	+	+
Increased merger and acquisition possibilities	+	
Handle the increased organization Scale	+	
Global Presence	+	
Cost reduction of development	+	
Reduction in time to market	+	+
Proximity to market	+	

TS Technological Support
NTS Non Technological Support

Table 8.1: Technological opportunities to exploit the benefits of GSD

of differences in language. It is possible to automatically translate the content of messages into the native language of all participants. However, these translate machines must be improved in order to be used on a regular basis. Other challenges which can be supported by technological aid are the lack of shared understanding between colleagues and the increased effort it takes to initiate contact with colleagues. Technology which makes it easier to get in touch with colleagues and technology which provide relevant project information to all team members, alleviate these challenges.

Now we have enumerated and examined the possibilities for technological support for all benefits and challenges we can apply the proposed classification to each benefit and challenge. In table 8.3 all the benefits and challenges which can be supported by technology are enumerated and the proposed classification is applied to each of them. In this table a double-plus sign indicates it is possible to use technological support from the corresponding category to directly exploit or alleviate a benefit or challenge respectively. A single-plus sign also indicates exploitation or alleviation is possible with the help of technological support from the corresponding category. However, either the technology does not alleviate the challenge directly, but it alleviates one or more consequences of the challenge, or the level of exploitation, which can be achieved by technology, is limited.

We will discuss the categorization of all benefits and challenges in order to give an overview of the possibilities technology offers, with respect to a certain benefit or challenge. In table 8.3 it can be seen that exploiting the benefit of being able to handle an increased product complexity, benefit B_1 can be supported by technologies offering features from categories R_3 and R_4 . Technological support which fulfills requirement R_3 reflects the current project status which is required to integrate the total product successfully, technological support which fulfills requirement R_4 causes the modules to be implemented according to their

8. TYPES OF TECHNOLOGY WHICH SUPPORT GSD

Challenge	NTS	TS
Lack of informal communication	+	+
Increased effort to initiate contact	+	+
Reduced hours of collaboration	+	+
Lack of shared understanding	+	+
Increased dependency on technology	+	
Increased complexity of the technical infrastructure	+	
Communication delay	+	+
Loss of cohesion	+	+
Reduced trust	+	+
Perceived threat from low-cost alternatives	+	
Increased team size	+	+
Differences in language	+	+
Differences in ethical values	+	
Differences in organizational vision	+	
Differences in managing individualism and collectivism	+	
Differences in terms of agreement	+	
Differences in time perception	+	
Differences in quality assessment	+	
Differences in design	+	
Differences in attitude toward hierarchy	+	

TS	Technological Support
NTS	Non Technological Support

Table 8.2: Technological opportunities to alleviate the challenges of GSD

specifications. Benefit B_2 can be supported by technology from categories R_1 , R_2 and R_3 , technology from category R_1 because it eases direct communication with the specialist, R_2 because it eases knowledge sharing in the entire development team and R_3 because it shows the status of the specialist. Benefit B_3 can be supported in a similar fashion as benefit B_2 because in both benefits collaboration between team members is very important. Direct communication, knowledge sharing and project transparency help team members to collaborate more effectively. The last benefit we discuss, benefit B_4 , can be supported from technology which fulfills requirement categories R_1 , R_2 and R_3 . Technology from category R_2 because it eases knowledge sharing in a distributed team which is working in different time-zones. Category R_3 is also useful because it reflects the current project status and so increases the transparency of the total project. Technology from category R_1 could also be beneficial because it allows team members to communicate directly and resolve issues at hand more quickly.

In the second part of table 8.3 all the challenges are classified. The first challenge, chal-

Benefit/Challenge	R1	R2	R3	R4
B1 Handle the increased product complexity			++	+
B2 Usage of specialized or skilled people	++	++	+	
B3 Access to a sufficiently large workforce	++	++	+	
B4 Reduction in time to market	+	++	++	
C1 Lack of informal communication	++	++		
C2 Increased effort to initiate contact	++		+	
C3 Reduced hours of collaboration	+	++	+	
C4 Lack of shared understanding	++	++	+	+
C5 Communication delay	++	+	+	
C6 Loss of cohesion	+	+	++	+
C7 Reduced trust	+		+	++
C8 Increased team size		++	++	
C9 Differences in language	++	++		

R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance

Table 8.3: Technological objectives to support GSD

challenge C_1 , can be alleviated by technological support which makes it easier for team members to communicate, category R_1 or technological support which eases knowledge sharing, category R_2 . Challenge C_2 can be alleviated by technology which facilitates direct communication and decreases the effort required to initiate contact, requirement category R_1 . Technology from category R_3 is also beneficial because it helps to determine the appropriate colleague to contact. Another challenge of GSD is that the hours of collaboration in a distributed development team are reduced. Technology which facilitate direct communication, knowledge sharing and transparency of the project status help to reduce the impact of this challenge. Challenge C_4 can also be supported by technological support from categories R_1 and R_2 , technological support from these categories directly increase the level of shared understanding. Requirements R_3 and R_4 alleviate the consequences of a lack of shared understanding among team members. By providing sufficient organizational knowledge and quality assurance procedures the impact of misunderstanding is reduced. The fifth challenge we discuss is similar to both challenge C_1 and challenge C_2 ; in order to reduce the communication delay one could use technologies which support direct communication and provide the availability of colleagues. Knowledge sharing can be used to deal with the consequences in the same manner as challenge C_4 . The loss of cohesion, challenge C_6 , can be alleviated from technology from all categories; technology from category R_3 directly alleviates this challenge by distributing details about the work activities of all team members among the whole team. Technology from the other categories do not alleviate the challenge directly and are indicated with a single plus in the table. The challenge of a reduced level

of trust, challenge C_7 , can directly be alleviated by technology which support quality assessment. The increased quality of the developed code causes the team members to trust in the capabilities of each other. Technology which facilitate direct communication or transparency of the project status help to reduce the impact of this challenge. Challenge C_8 can only be alleviated when team members have a good understanding of both *organizational* and *technical* project knowledge. Finally challenge C_9 can be alleviated by using an *universal translator*. This kind of technology is especially useful when it is used to facilitate direct communication or knowledge sharing between team members who do not speak the same language.

Now we have applied the proposed classification to all benefits and challenges, that can be supported by means of technology, it appears that each of the requirement categories support at least a single benefit or challenge and it is possible to categorize each benefit and challenge into at least one of the requirement categories. This classification can be used to determine which kind of technological support is needed in order to support GSD in general.

8.2 Types of technology which support agile GSD

In order to be able to discuss the kind of technology useful when incorporating agile aspects into a global development approach, a categorization for the different types of technological support is needed. In the previous section a categorization for technological aid to help with global software development in general is presented. This categorization is largely applicable to agile GSD as well, because of the large similarity between both the concept of GSD in general and the concept of a GSD approach with incorporated agile aspects. This large similarity originates from the nature of both concepts; in both development approaches the collaboration between people is complicated because of the three distances faced in a globally distributed setting. Because of the large similarity, we will start with the requirement categories defined in the previous section, followed by discussing whether this list of requirement categories is both applicable and complete with respect to agile GSD as well.

Requirement R_1 ; *Facilitate direct contact between colleagues*, is applicable since the importance of direct contact between colleagues is still present. In fact, it has only grown, especially with the incorporation of aspect A_1 : *Close collaboration among the members of the development team*. The same is true both for requirement R_2 ; *Facilitate knowledge sharing among colleagues* and requirement R_3 , *Facilitate transparency of the project status*, as these requirement mainly aim to alleviate the difficulty of collaboration between people and in agile development close interpersonal collaboration is essential. Lastly, requirement R_4 , *Facilitate quality assurance*, is applicable as well since more valuable feedback offers the same benefits with respect to agile GSD as it does with GSD in general. If anything, it is even more important because the agile development approach explicitly recognizes the necessity of elaborate feedback.

Having discussed the categories used to categorize the technological support for GSD in

general and showing these categories all apply in the case of agile GSD as well; we should determine whether this list of features completely covers all that can be supported with technology. We feel that this is not the case since none of these requirements explicitly help the continuous integration, and thus the possibility of frequent builds, of the system. So, we propose to use the list of requirement categories defined in the previous section extended with a fifth one, which consists of technological support which facilitates continuous integration and frequent builds. The complete list of requirement categories applicable to agile GSD is as follows:

R₁ Facilitate direct contact between colleagues

Technological support which facilitates direct communication between two or more actors.

R₂ Facilitate knowledge sharing among colleagues

Technological support which facilitates the sharing of *technical* project knowledge.

R₃ Facilitate transparency of the project status

Technological support which facilitates the sharing of *organizational* project knowledge.

R₄ Facilitate quality assurance

Technological support which facilitates quality assurance functions to monitor and guarantee the quality of the product.

R₅ Facilitate continuous integration and frequent builds

Technological support which eases the process of continuously integrating the system as well as producing builds frequently.

Chapter 9

How the different types of technological support are applicable to support agile GSD

In order to incorporate the five aspects of the agile development approach that influence distance into the GSD development process in the best way possible, the challenges faced should be alleviated and the benefits should be exploited as much as possible. One way to achieve this is by making use of technological support. In this chapter, we will discuss what types of technological support are beneficial to which aspects or more precisely: which challenges and benefits belonging to the aspects can be supported by which type of technological aid. We will start, in the first section, by describing a structured approach to derive this for each aspect. In this derivation first the relation between the aspects and the distances is discussed followed by a discussion how each of the challenges and benefits is best supported by the different types of technological support. Subsequently, in the second section, we will apply this approach consecutively for each aspect to derive how they are best supported by technology. In the third and final section the relations between all aspects and distances are summarized as well as what types of technological support are beneficial to which aspects. The aim of this is to reflect on how technologically supporting the incorporation of agile aspects into the GSD development process impacts the overall interdependencies between the aspects, distances and all corresponding challenges and benefits.

9.1 An approach to derive technological support for agile GSD

In this section we will describe a structured approach to derive which challenges and benefits belonging to each of the five aspects of agile development that influence distance can be supported by which type of technological aid. Firstly for every aspect, the relationship between it and the distances caused by working globally distributed will be illustrated. This is done by creating a graph with the distances and aspects depicted as nodes and the interdependencies as directed arcs, which correspond to either a benefit or a challenge. The arcs which correspond to a benefit are directed from the aspect to the distance it targets and

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

represent how the aspect helps to cope with the target distance. The arcs which correspond to a challenge are directed from the distance which causes the challenge to the aspect the challenge opposes and represent how the distance makes it harder to carry out the aspects. Following this, we will discuss which benefits can be supported and which challenges can be alleviated by means of technological support. We will do this by discussing support from which of the categories of technological support is beneficial with respect to the benefit or challenge under discussion. We will conclude the discussion of each aspect with a table summarizing technological support from which categories is beneficial with respect to each of the benefits and challenges discussed. In this table a double-plus sign indicates it is possible to use technological support from the corresponding category to directly exploit or alleviate a benefit or challenge respectively. A single-plus sign also indicates exploitation or alleviation is possible with the help of technological support from the corresponding category. However, either the technology does not alleviate the challenge itself or the level of exploitation of the benefit, the technology can achieve, is limited. The final row of such tables will consist of a summarization in which the overall use of the various categories of technological support with respect to the current aspect is displayed. This row is obtained by picking the highest scoring value from the column for each cell. This is done in this fashion as opposed to, for instance, averaging the result, because we did not compare the challenges and benefits to each other, to produce an overall weight allocation. This way the final row of the table will show which categories of technological support can be used for direct or indirect support of at least one benefit or challenge, for each aspect.

9.2 How each aspect can be supported by the different types of technology

A₁ Close collaboration among the members of the development team

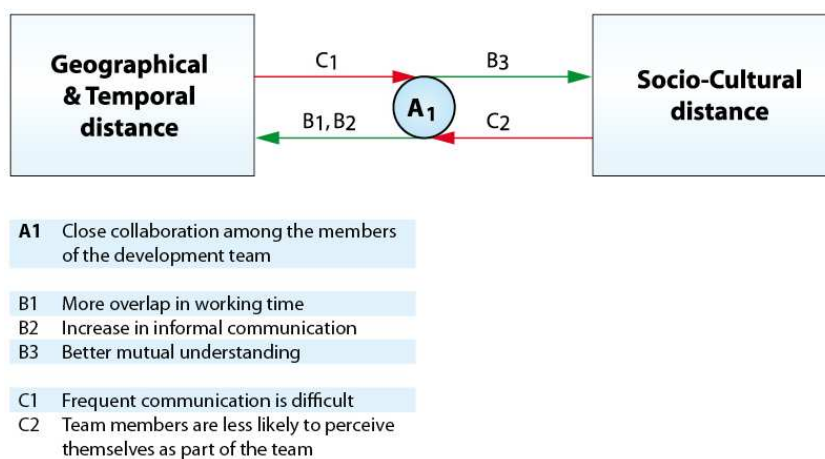


Figure 9.1: Relationships between aspect A₁ and the distances faced in GSD

The main challenge with respect to aspect A_1 is challenge C_1 : the fact that frequent communication is difficult in a distributed setting. Because of this, close collaboration among team members is more difficult. This can be alleviated by technological support, which makes it easier for team members to communicate and by technological support which eases knowledge sharing in the entire development team; categories R_1 and R_2 respectively. Features belonging in category R_3 could also help to initiate contact, if the specific feature makes it possible to determine who to contact by offering information with respect to individual knowledge and availability of colleagues. From figure 9.1 it can be gathered that when the influence of challenge C_1 is decreased, benefits B_1 , B_2 and B_3 are better exploited and in turn the influence of challenge C_2 is decreased as well. This derivation also works the other way around: technological support for benefit B_2 will decrease the geographical and temporal distance which in turn decreases the magnitude of challenge C_1 . In this particular example, the same types of technological support that help to deal with challenge C_1 also support benefit B_2 because C_1 and B_2 are closely related. So, when incorporating aspect A_1 into a GSD project technological support offering features from category R_1 and R_2 are most important and features from category R_3 could also be beneficial. This is summarized in table 9.1.

Challenge/Benefit	R_1	R_2	R_3	R_4	R_5
C_1	++	++	+		
B_2	++	++	+		
Aspect 1	++	++	+		

C_1	Frequent communication is difficult
B_2	Increase in informal communication
R_1	Facilitate direct contact between colleagues
R_2	Facilitate knowledge sharing among colleagues
R_3	Facilitate transparency of the project status
R_4	Facilitate quality assurance
R_5	Facilitate continuous integration and frequent builds

Table 9.1: Overview of technological support for aspect A_1

A_2 Short iterations, frequent builds and continuous integration

With respect to this aspect various challenges and benefits can be alleviated or exploited respectively, by means of technological support. For one, challenge C_5 can be alleviated both by facilitating knowledge sharing and facilitating transparency of the project status. This is because knowledge sharing will provide team members with a better and more complete understanding of the technical project knowledge: the knowledge required to actually solve the difficulties associated with the project. The transparency of the project status, on the other hand, will ease the access to organizational knowledge, which is the knowledge

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

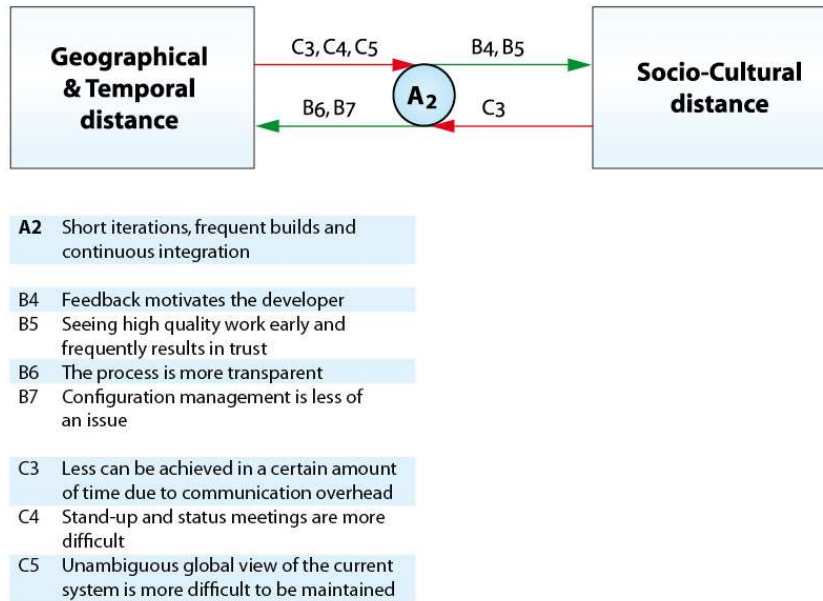


Figure 9.2: Relationships between aspect A_2 and the distances faced in GSD

regarding how the project is to be carried out by the development team and how this is progressing. Both these types of information together make up an unambiguous global view of the current system. Technological support from categories R_2 and R_3 alleviate access to these types of information and thus alleviate challenge C_5 .

Another challenge which can be alleviated by means of technological support is challenge C_4 . By facilitating direct contact, having meetings geographically distributed is made easier. Therefore technological support which offers features from category R_1 will directly target this challenge. Another large factor when dealing with this challenge is the difficulty to arrange the meetings when the people required to be present do not share the same working hours. By facilitating access to the information regarding the availability of the participants, as well as other relevant data with respect to the organizational part of the project, this challenge can be directly dealt with. Technological support from category R_3 accomplishes this. Lastly, technological support from category R_2 will also help dealing with this challenge, although in a less direct manner, because when knowledge sharing is improved, the actual need for having meetings is decreased.

The last challenge associated with this aspect, challenge C_3 , can also be supported by technology. Technology from category R_1 , R_2 and R_3 can all provide this support because they all facilitate a certain aspect of communication and when communication is less difficult, the communication overhead will be decreased.

Next we will discuss how the benefits caused by this aspect can be exploited by means

of technological support. Firstly benefit B_4 , can be further exploited by facilitating quality assurance, technology from category R_4 , because better quality assurance will improve the quality of the feedback. Technological support from categories R_1 , R_2 and R_5 help to exploit this benefit in an indirect manner. Support from R_5 causes the feedback to be more up-to-date because, the information the feedback is based on is created more often. Support from R_1 and R_2 ; finally, help exploit this benefit by helping to make the feedback accessible to the appropriate people. Benefit B_5 is related to benefit B_4 in the sense that feedback can also concern the quality of work. In this benefit however, emphasis lies on team members witnessing the high quality themselves. This can be achieved, either by examining a prototype of the system or by taking note of certain measures which increase the chances of creating a high quality product. An example of such a measure is employing a certain testing scheme. When this is done and the results of these tests are positive, this is also an indicator that the work is of high quality. These ways to witness the high quality of work can be supported by technology offering features from categories R_4 and R_5 . Technology from R_4 offers measures to assure a high quality system is created and because of support from R_5 it is easier to have access to a prototype of the current system.

Another benefit caused by aspect A_2 which can be further exploited by using technological support is benefit B_6 . For one, technology from category R_3 provides an overall view of the current state of the project and thus increases the transparency of the project status directly. Technology from R_4 and R_5 is also useful since R_4 enables the quality aspect of the progress to be available and R_5 enables the overview to be updated more often and thus to more accurately reflect the actual state of the project.

The last benefit of aspect A_2 , benefit B_7 , can also be supported by technology. This is because continuous integration eases the problem of configuration management, which is the management of the evolution of the complete system [138]. This is done by performing the integration before the entire system is complete on the portion of the system that is complete. Because of these 'so-called' partial system integrations, problems are detected earlier and so can be more easily dealt with. This process can be supported by technology by making these frequent partial system integrations easier to do. This is achieved by technology which fulfills requirement R_5 .

A_3 Decentralizing the decision making

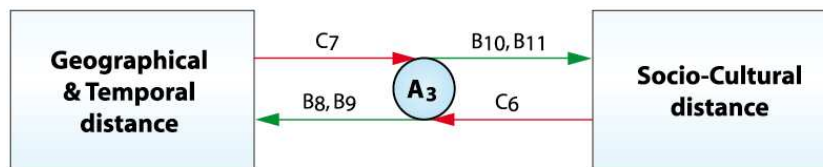
With respect to this aspect challenge C_7 can be alleviated by use of technological support by easing the access to the knowledge necessary to reach the right decisions. Because this mainly concerns technical decisions, technological support from R_2 can be used to directly alleviate the challenge, while support from R_3 is useful as well, yet in an indirect manner. This is because knowledge regarding the organizational status and setup of the project will provide a context to make the decisions in, yet the information the decisions are mainly based on will be technical information. The same is true with respect to the exploitation of benefit B_8 since when better decision are made, less control is needed in both directions be-

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

Challenge/Benefit	R1	R2	R3	R4	R5
C3	++	++	++		
C4	++	+	++		
C5		++	++		
B4	+	+		++	+
B5				++	++
B6			++	+	+
B7					++
Aspect 2	++	++	++	++	++

C3	Less can be achieved in a certain amount of time due to communication overhead
C4	Stand-up and status meetings are more difficult
C5	Unambiguous global view of the current system is more difficult to be maintained
B4	Feedback motivates the developer
B5	Seeing high quality work early and frequently results in trust
B6	The process is more transparent
B7	Configuration management is less of an issue
R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 9.2: Overview of technological support for aspect A_2



A3	Decentralizing the decision making
B8	Less control needed
B9	Better progress estimations
B10	Autonomy motivates the developers
B11	Being trusted results in trust
C6	Some developers might not be autonomous and independent enough to make decisions themselves
C7	Having access to sufficient information in order to reach correct decisions is more difficult

Figure 9.3: Relationships between aspect A_3 and the distances faced in GSD

tween the people technically implementing the project and the people managing the project. Benefit B_9 is further exploited directly by technological support from both R_2 and R_3 . This is because both knowledge regarding the content of work to be performed and the context this work is to be performed in, directly influences the estimation of the current and expected progress.

Again, like with aspect A_1 , some benefits and challenges cannot be directly supported by technological aids. These benefits and challenges are however also supported indirectly because the technological support for the other benefits and challenges cause the entire aspect to function better.

Challenge/Benefit	R_1	R_2	R_3	R_4	R_5
C7		++	+		
B8		++	+		
B9		++	++		
Aspect 3		++	++		

C7	Having access to sufficient information in order to reach correct decisions is more difficult
B8	Less control needed
B9	Better progress estimations
R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 9.3: Overview of technological support for aspect A_3

A_4 Customer involvement

With respect to this aspect challenge C_8 can be alleviated in a similar fashion as challenge C_1 from aspect A_1 . Thus technological support from category R_1 and R_2 support it directly because this eases communication and R_3 supports it indirectly since this only helps with the arrangement of direct contact. Benefit B_{12} can be further exploited by facilitating for constant integration and frequent builds because this will provide the customer with prototypes, to give feedback on, more often. Because of this the customer can be involved without actually having to be on site, since the prototype can be sent to a remote location. This is also why technical support from categories R_1 and R_2 also help further exploitation of this benefit, since if both the feedback and the information regarding and including the prototype are as easily accessible to the customer as possible; this will result in the best and most feedback possible.

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

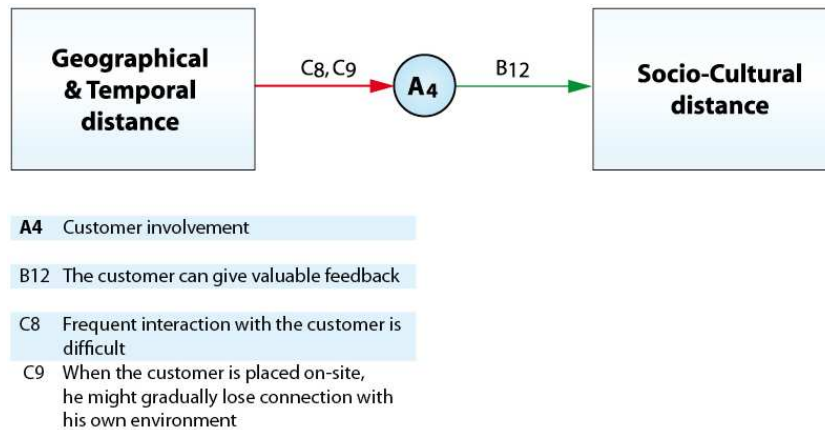


Figure 9.4: Relationships between aspect A_4 and the distances faced in GSD

What is noticeable in figure 9.4 is the absence of loops. Because of this the challenge that cannot be directly alleviated by technological support, challenge C_9 , also is not indirectly alleviated within this aspect. For obvious reasons when other aspects are incorporated that do influence the geographical and temporal distance in a positive manner, this *will* alleviate challenge C_9 .

Challenge/Benefit	R_1	R_2	R_3	R_4	R_5
C8	++	++			
B12	+	+			++
Aspect 4	++	++			++

C8	Frequent interaction with the customer is difficult
B12	The customer can give valuable feedback
R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 9.4: Overview of technological support for aspect A_4

A_5 Collective ownership of work

Aspect A_5 can only be supported by technological support which alleviates challenge C_{11} . This can be done by using technological support from category R_2 , because support from this category eases access to technological project information and so also to all work pro-

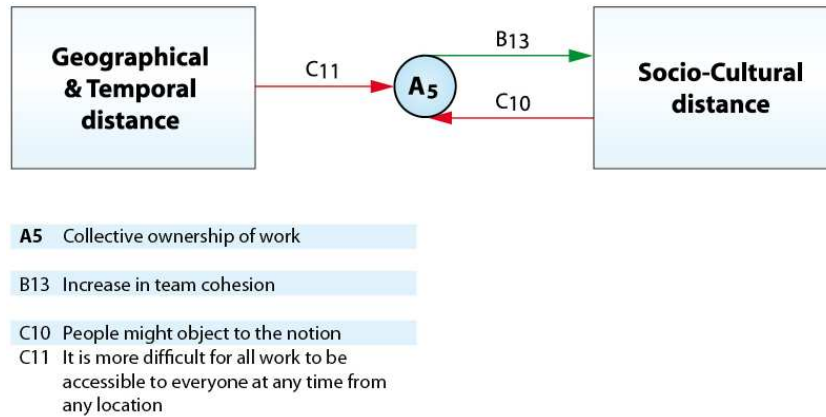


Figure 9.5: Relationships between aspect A_5 and the distances faced in GSD

duced by the development team. Technological support from category R_3 can support challenge C_{11} in an indirect fashion because working together in a single work-base can be done more effectively and efficiently when knowledge, regarding the current and past activities of the other members of the team, is available.

In figure 9.5 the absence of a loop on the left-hand side is most noticeable. However, in contrast to the previous aspect, here all benefits of the aspects can be exploited and all challenges can be alleviated, either directly or indirectly, within this aspect. This is because challenge C_{11} on the left-hand side can be directly alleviated by technological support as discussed in the previous paragraph. When this is done this will improve the overall functioning of the aspect as a whole and thus indirectly further help to exploit benefit B_{13} . This will result in a decrease of the social-cultural distance and thus will also put a strain on challenge C_{10} .

Challenge/Benefit	R_1	R_2	R_3	R_4	R_5
C11		++	+		
Aspect 5		++	+		

C11	It is more difficult for all work to be accessible to everyone at any time from any location
R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 9.5: Overview of technological support for aspect A_5

9.3 Overview

In this section we will provide overviews of the material discussed in this chapter. First we will provide an overview of the relationships that exist between the distances and the aspects in the same way as was done in the figures in the previous section, in figure 9.6. Next in figure 9.7, the same figure is presented, however with all relationships that cannot be supported by technological aid removed. Following this two tables, table 9.6 and 9.7, are depicted. Table 9.6 depicts by which types of technological support the corresponding benefits and challenges can be supported. Table 9.7 shows a summary of the same information to make clear which types of technological support can be used to support which aspects.

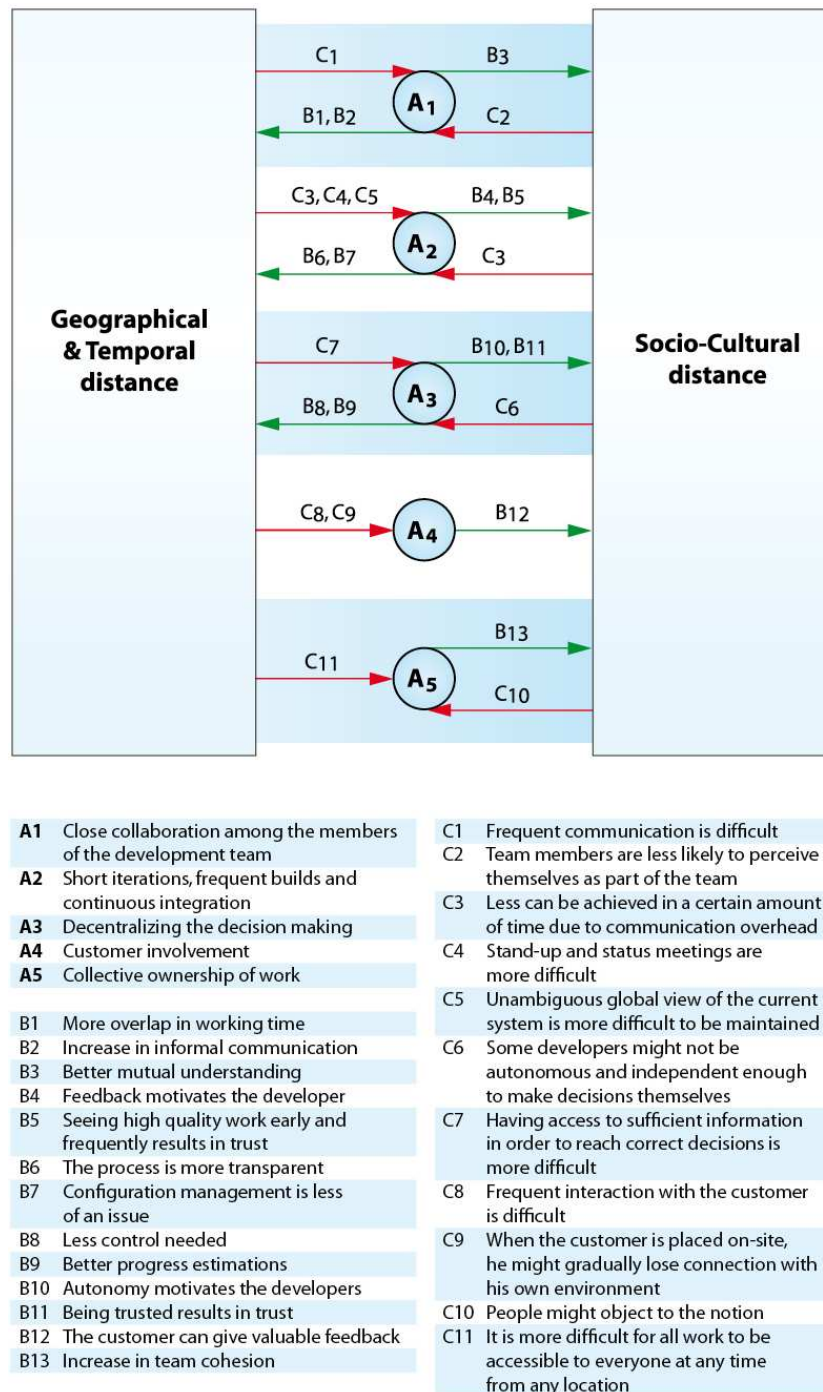


Figure 9.6: Complete overview of relationships aspects and distances

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

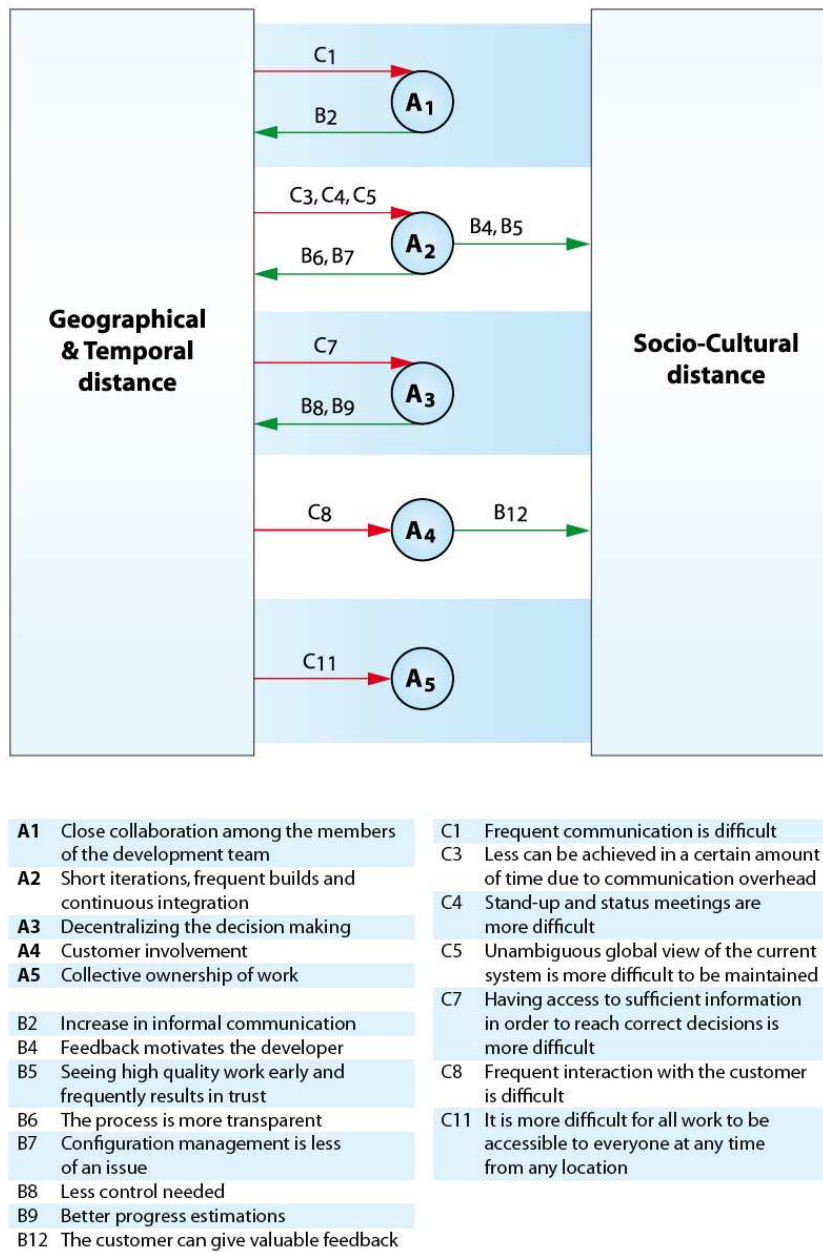


Figure 9.7: Overview of the relationships between aspects and distances that can be supported by technological support

Aspect	Challenge/Benefit	R1	R2	R3	R4	R5
A1	C1	++	++	+		
	B2	++	++	+		
A2	C3	++	++	++		
	C4	++	+	++		
	C5		++	++		
	B4	+	+		++	+
	B5				++	++
	B6			++	+	+
	B7					++
A3	C7		++	+		
	B8		++	+		
	B9		++	++		
A4	C8	++	++			
	B12	+	+			++
A5	C11		++	+		

A1 Close collaboration among the members of the development team

A2 Short iterations, frequent builds and continuous integration

A3 Decentralizing the decision making

A4 Customer involvement

A5 Collective ownership of work

R1 Facilitate direct contact between colleagues

R2 Facilitate knowledge sharing among colleagues

R3 Facilitate transparency of the project status

R4 Facilitate quality assurance

R5 Facilitate continuous integration and frequent builds

B2 Increase in informal communication

B4 Feedback motivates the developer

B5 Seeing high quality work early and frequently results in trust

B6 The process is more transparent

B7 Configuration management is less of an issue

B8 Less control needed

B9 Better progress estimations

B12 The customer can give valuable feedback

C1 Frequent communication is difficult

C3 Less can be achieved in a certain amount of time due to communication overhead

C4 Stand-up and status meetings are more difficult

C5 Unambiguous global view of the current system is more difficult to be maintained

C7 Having access to sufficient information in order to reach correct decisions is more difficult

C8 Frequent interaction with the customer is difficult

C11 It is more difficult for all work to be accessible to everyone at any time from any location

Table 9.6: Overview of technological support for all benefits and challenges

9. HOW THE DIFFERENT TYPES OF TECHNOLOGICAL SUPPORT ARE APPLICABLE TO SUPPORT AGILE GSD

<i>Aspect</i>	<i>R1</i>	<i>R2</i>	<i>R3</i>	<i>R4</i>	<i>R5</i>
A1	++	++	+		
A2	++	++	++	++	++
A3		++	++		
A4	++	++			++
A5		++	+		

A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work
R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 9.7: Overview of technological support for all aspects

Chapter 10

How to support the incorporation of agile aspects into the GSD process with technology

The previous chapter was concluded by giving an overview of supporting technology from which requirement categories are useful with respect to each challenge and benefit associated with incorporating agile aspects into the GSD process. In this chapter we will start with drawing some general conclusions from this overview. Following this we will discuss specific technologies supporting collaborative development. Here we will discuss communication related technologies, software development related technologies and how these solution support the various challenges and benefits identified in the previous chapter. We will conclude this chapter by discussing how integrating all the different types of technological support is beneficial.

10.1 Communication versus Software Development related technology

As announced in the introduction, in this section we will draw conclusions, regarding the importance of the different forms of technological support, from the overviews presented in table 9.6 and table 9.7. The first thing to mention is that which specific forms of technological support are most beneficial depends on the specific project for which it is attempted to derive this. It is for instance possible to not incorporate all aspects of agile software development, in which case part of the table does not apply. Next to this, all challenges and benefits associated with the aspects, elected to be incorporated into the project, will have varying degrees of importance in different projects. Because of this, the challenges and benefits will have different prioritizations in different projects and drawing conclusions for the general case is difficult.

Despite the inherent difficulties with drawing general conclusions, still some conclusions can be drawn. For one, it can be seen in table 9.6 that requirement category R_2 supports

10. HOW TO SUPPORT THE INCORPORATION OF AGILE ASPECTS INTO THE GSD PROCESS WITH TECHNOLOGY

all five aspects directly. So, in general, technological support from category R_2 will improve an agile GSD project. Another thing to take note off in the table, is that technological support from categories R_4 and R_5 can only be used to exploit benefits and not to alleviate challenges, caused by working globally distributed. In fact, requirement categories R_4 and R_5 are very much related with respect to their impact on an agile GSD project. This can be derived more explicitly by defining a fifteen-entry tertiary vector for each of the requirement categories; the columns in table 9.6. Then, for each of the ten possible combinations of two vectors the vectors are compared and the absolute difference in the number of '+'-signs is summated. The overview of this, sorted from most similar to most different requirement categories, is shown in table 10.1.

Combination	Difference
R4 - R5	5
R1 - R2	11
R2 - R3	12
R1 - R3	15
R1 - R4	15
R1 - R5	16
R3 - R4	18
R3 - R5	21
R2 - R4	24
R2 - R5	25

R1	Facilitate direct contact between colleagues
R2	Facilitate knowledge sharing among colleagues
R3	Facilitate transparency of the project status
R4	Facilitate quality assurance
R5	Facilitate continuous integration and frequent builds

Table 10.1: Relatedness of requirement categories

As we mentioned, requirement categories R_4 and R_5 are very much related with respect to their impact on the agile GSD process. Next to this R_1 and R_2 are also quite related. When the division of the requirement categories into the two sets $\{R_1, R_2\}$ and $\{R_4, R_5\}$ is made it is quite clear R_3 is most related with the first one since the difference between R_3 and both entries of this set is smaller than the difference between R_3 and both the entries of the other set. Following this reasoning we propose to distinguish the list of requirement categories into two sets of categories:

$$Set_1 \{R_1, R_2, R_3\}$$

$$Set_2 \{R_4, R_5\}$$

Both these combinations of related categories make sense since R_1 , R_2 and R_3 all aim to support the actual communication between developers and R_4 and R_5 aim more at support-

ing software development related matters. Another reason why this division is accurate is that the maximum cut also divides the requirement categories in these two groups. In the next section we will separately discuss the technological realization of both the groups of requirement categories defined here.

10.2 Collaborative technologies

In this section specific technology supporting collaboration development will be discussed. Firstly the communication related technologies will be discussed and secondly the software development related. The discussion of both these sections, however, differs. On the one hand the discussion of the communication related technologies is structured by the type of communication provided by the different technologies discussed. On the other hand the discussion regarding the software development related technologies is structured by the goal of the discussed technologies. This is done in this fashion because the software development related technologies are defined with a specific goal in mind in the context of collaborative development. The different communication related technologies, however, all share a common goal: to ease the exchange of information between people. Even though all forms of communication share one abstract common goal, some forms of communication are more appropriate to reach certain goals than others. An example is using an asynchronous form of communication like email or a forum and attempt to have a synchronous discussion; constantly waiting for the other to answer. This is a feasible way to communicate but there exist other, more appropriate solutions like instant messaging software.

10.2.1 Communication related technologies

To be able to discuss the communication related technologies in a structured fashion we will define two characteristics of such technologies. Firstly communication can either be '*synchronous*' or '*asynchronous*'. Communication is regarded '*synchronous*' when the sending and receipt of messages between actors communicating can be regarded as instantaneous. When the sending and receipt of messages cannot be regarded as instantaneous the communication is called '*asynchronous*'. An example of the difference between '*synchronous*' and '*asynchronous*' communication is that using an instant messenger client to ask someone, available at his computer, a question is '*synchronous*' while emailing the same person a question is regarded '*asynchronous*'. The second characteristic of communication related technologies we defined is that it can either be '*direct*' or '*indirect*'. Communication can be considered '*direct*' when the intended recipient or recipients of the communication are known in advance. When this is unknown the communication is considered '*indirect*'. An example of the difference between '*direct*' and '*indirect*' communication is that making knowledge available on a Wiki for all developers is '*indirect*' whereas sending an email to a specific group of people is '*direct*'. In the remainder of this section we will first discuss specific technologies which support the different types of communication followed by an overview of the discussed technologies. We will conclude with discussing how these types of communication are related to the requirement categories and how the discussed technolo-

gies are best applied to support the challenges and benefits associated with the incorporation of agile aspects into GSD.

Synchronous Communication

In this section we will discuss examples of well-known technologies which can be used to facilitate synchronous communication. These technologies will be discussed by the type of information they are able to transmit. Firstly there are technologies facilitating synchronous '*text-based communication*'. Instant messaging software, such as AIM¹, Yahoo Messenger² and MSN Messenger³, is such a technology as it creates the possibility of real-time text-based communication between two or more participants. It is a technology facilitating direct communication; the intended recipient or recipients of the communication should be known in advance. There are also technologies facilitating '*text-based communication*' which are indirect. Such technologies are known as '*chat-rooms*' in which the users subscribe to a number of rooms or channels. Everyone subscribed to a certain room or channel is able to send a message to all users currently subscribed to it. Examples of such technologies are web-based chat-rooms and IRC.

Secondly there exist technologies facilitating synchronous '*audio- and video-based communication*'. Again, a distinction can be made between technology supporting direct and indirect communication. For direct communication technologies exist to send audio or video directly to the recipient or recipients. Such technologies are commonly known as conferencing tools and examples are applications implementing VOIP, telephones and video-phones [59]. For indirect communication supporting technologies of this kind are different types of broadcasting systems. An example of this is an intercom system, which transmits audio signals and delivers it to every development location.

Thirdly it is possible to use a shared virtual space where communication via different media is controlled by movement within this virtual space [63]. Such a technology attempts to mimic face-to-face communication as close as possible by simulating the environment of a normal meeting. Examples of such technologies are I-maginer⁴ and Workspace 3D⁵.

Finally there exist synchronous communication technologies facilitating remote control and inspection; so-called application sharing technologies. Such technologies can be used to remotely inspect each others work but also to collaboratively work on a common goal or demonstrate something. They are intended for supporting direct communication as the intended recipient is known in advance. An example of such a technology is screen sharing software, with or without the capability of remote control, like Virtual Network Comput-

¹<http://www.aim.com>

²<http://messenger.yahoo.com>

³<http://messenger.live.com>

⁴<http://www.i-maginer.fr>

⁵<http://www.tixeo.com>

ing [127] and Team Viewer⁶. A second example is a virtual whiteboard, either physical or non-physical, which can be used to remotely visualize ideas to each other for example for informal workgroup meetings [113].

Asynchronous Communication

In this section we will discuss examples of well-known technologies which can be used to facilitate asynchronous communication. Firstly there exist technologies which facilitate direct communication. Again these can be text-based like SMS [114, 20] and email [118], audio-based like voicemail and composite-based like fax, file transfer and MMS [20].

Secondly there exist technologies which facilitate indirect communication. Since the technologies which support this type of communication support most types of information, the subdivision we used to structure the previous categories is inappropriate. For this type of communication we will structure the supporting technologies using the type of content of the communication they provide. The first of such types is technical project knowledge: *the knowledge required to actually solve the difficulties associated with the project*. An example of technology supporting the communication of technical project knowledge is a code-repository like Subversion (SVN) [42] or Concurrent Versions System (CVS) [65]. The second type of content of communication regards organizational project knowledge: *the knowledge regarding how the project is to be carried out by the development team and how this is progressing*. Examples of this are scheduling software, like an online agenda, tracking tools like Bugzilla⁷ and Trac⁸ and tools which display a group of people and their availability, like many instant messengers provide by means of a buddy list. Lastly there are several technologies in this category which can be used for communicating both technical and organizational project knowledge. Examples of this are forums, wikis, newsgroups, mailing lists, blogs and RSS-feeds [68].

Overview

In figure 10.1 an overview of the communication related technologies is given structured using the type of communication they support.

Relation between types of communication and communication related requirement categories

In this section the relation between the types of communication defined above and the communication related requirement categories (R_1 , R_2 and R_3) will be discussed. Since R_1 , R_2 and R_3 all have to do with communication these requirements can be fulfilled by each of the types of communication. Some of these, however, are more appropriate to use in certain situations than others. For category R_1 it is necessary to be able to discuss matters with colleagues and notify them of certain matters. The most suitable type of communication

⁶<http://www.teamviewer.com>

⁷<http://www.bugzilla.org>

⁸<http://trac.edgewall.org>

10. HOW TO SUPPORT THE INCORPORATION OF AGILE ASPECTS INTO THE GSD PROCESS WITH TECHNOLOGY

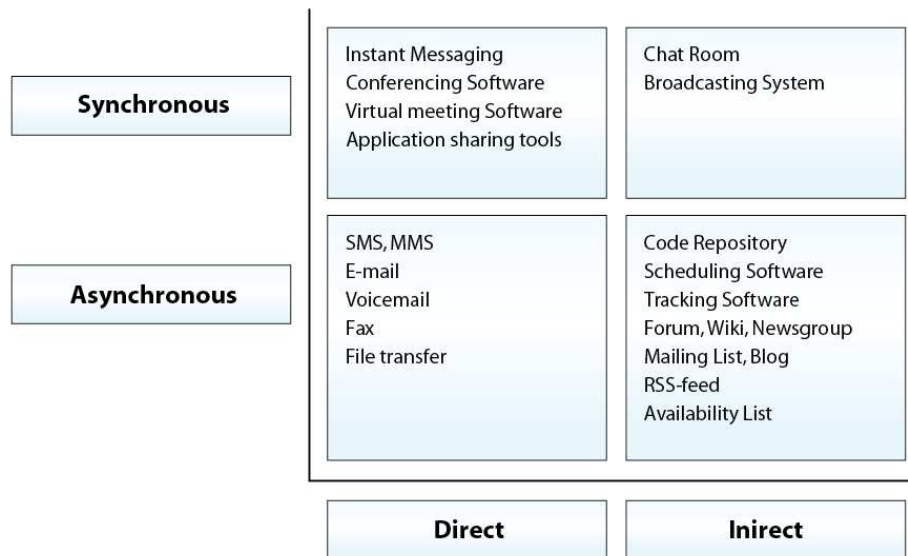


Figure 10.1: Communication related technologies

to achieve this is synchronous communication. Both direct and indirect communication are of equal importance since both communication types provide for immediate feedback and hence have the ability to help reach immediate decisions. They only differ with respect to the fact if it is known in advance with whom to converse. The best way to support these forms of synchronous communication is the technologies which directly supports them. For instance direct synchronous communication is supported well by instant messengers and indirect synchronous communication is supported well by chat-rooms. When, however, such means of communication are either unavailable or inapplicable due to certain project properties, such as temporal dispersion, asynchronous technologies can also be useful. Examples of this are the exchange of emails (direct communication) and the discussion of certain matters on a forum (indirect communication). Next, requirement categories R_2 and R_3 will be jointly discussed since similar restrictions with respect to the type communication which facilitate them, apply. This is because R_2 and R_3 both regard the sharing of information among colleagues. They only differ with respect to the content of the information being shared. R_2 and R_3 differ with R_1 with respect to that the sender does not intend for the communication to reach the recipient immediately. This is for instance the case when the sender wants to make documentation of a certain functionality available. This is best achieved by technology supporting asynchronous communication like for instance a wiki. Synchronous communication technologies can also be used since it is possible to access synchronously received information in an asynchronous fashion. An example of this is when a message is received via an instant messenger, the message can stay on the screen for a long period of time or it can be logged to a text file. Therefore the recipient can read the message when it is needed. Finally, like for R_1 , whether direct or indirect communication is required, relies solely on the fact whether it is known in advance for whom the communication is intended.

Which communication related technologies are best equipped to support the agile aspects

Which specific technology is best equipped to either exploit or alleviate a specific benefit or challenge depends on the type of information that needs to be communicated. For instance to deal with C_4 , *standup and status meetings are more difficult*, a synchronous communication technology which is as rich as possible, like video or virtual-based conferencing technology, is most suitable. This is because such technologies transfer information which contains a high content of tacit knowledge about other participants, in the form of facial expressions, verbalizations, movements or postures which helps to better understand one-another [100]. When aiming at making frequent communication between colleagues, distributed in time and space, possible, rich communication with respect to tacit knowledge again can be beneficial. This particularly regards situations where either one colleague is explaining something to another, colleagues are having a mutual discussion or a member of the development team is discussing something with the customer. When the colleagues, however, are attempting to actually develop together, for example when they are *pair programming* from XP, tools facilitating remote control and inspection combined with more simple synchronous communication technologies like an instant messaging program or an audio connection are more useful. This is because in this particular case having the same view of what is being worked on is more important than tacit knowledge gathered from for example the facial expression of the other. Less direct collaborative development, such as collaboratively developing software in a development team, requires other kinds of technological support. When the developers are not working together as closely as in *pair programming* they still need to share information to work together effectively. This information can be roughly divided in two types of data; the actual production code, for which a code repository is the most suitable supporting communication technology and technical meta-data about the system, for which technical solutions like wikis, forums and even email are appropriate. Next to the actual work, information regarding the project and the development team itself is also important. Examples of tools which are suitable for this are scheduling software and tracking software like BugTracker and Trac.

In this section we mentioned a variety of technologies which are able to facilitate communication. In order to support the communication related benefits and challenges associated with the agile aspects, appropriate supporting technologies should be selected based on both the type of communication and the type of information that needs to be communicated.

10.2.2 Software development related technologies

In section 10.1 we derived that requirement categories R_4 and R_5 regard software development related technologies. Because software development related technologies in the context of collaborative development are all created with a certain goal in mind we will use the division of these technologies in those that support *quality assurance* (R_4) and those that support *continuous integration and frequent builds* (R_5) to structure this section. Because of this the technologies which are discussed in discussion are assigned to the requirement

10. HOW TO SUPPORT THE INCORPORATION OF AGILE ASPECTS INTO THE GSD PROCESS WITH TECHNOLOGY

categories and thus no specific section to discuss this assignment is needed. The section is concluded with a discussion of how the discussed technologies are best applied to support the challenges and benefits associated with the incorporation of agile aspects into GSD.

Quality assurance

In this section technological support which facilitates quality assurance will be discussed. Firstly the quality of a software product can be guaranteed testing the behavior of the system. This can be supported by technology which allows to automate the testing of functional requirements. Examples of such technologies are tools that automate white-box [28] testing like Selenium⁹ and Watir¹⁰ and tools that automate black box [28] testing like JUnit¹¹ and TestNG¹². Tools that support automated white box testing often also assist with tracing the fault to the actual code segment where it occurs. Next to functional requirements, non-functional requirements, related to for example performance and load capacities, can be tested automatically. Examples of tools which support this are Apache JMeter¹³ and OpenSTA¹⁴.

Writing the tests for the test automation tools discussed above can be quite labor intensive. Therefore tools that provide automatic test case generation can help. Galler et al. [61] provide an overview of tools which provide automatic test generation for Java. Most of these tools automatically generate test cases based either randomly or based on explicit specifications of the classes. Some tools, like TestEra [98], Korat [25] and QuickCheck [34], test the various methods in a class in isolation while a tool like T2 [119] test also test the interrelated functionality of the methods. Once the test cases are generated it is possible to test for internal errors, runtime exceptions and violations of the specification.

Besides testing whether the system satisfies the requirements the quality of the system can also be assured by the direct elicitation of errors in the source code. Findbug¹⁵, for example, is a program which searches the source code for known errors like null-pointer dereferencing and reports any problem it discovers. Problems in the source code can also be discovered by determining which sections of code are more likely to contain a fault. This can be done by analyzing the source code using so-called code metrics, which are values which quantify how well a certain section of code is written [17]. An example of bad section of code is using a deep loop-nesting structure. Besides making faults more likely in a particular section of code, a bad metric score also indicates the section of code is harder to read and maintain. Hence, even when such a section of code does not contain an actual fault, the quality of the overall product will increase by refactoring the code to improve the metrics. Examples of

⁹<http://seleniumhq.org>

¹⁰<http://wtr.rubyforge.org>

¹¹<http://www.junit.org>

¹²<http://testng.org>

¹³<http://jakarta.apache.org/jmeter>

¹⁴<http://www.opensta.org>

¹⁵<http://findbugs.sourceforge.net>

tools which support quality assurance by means of metrics are: PMD¹⁶, Metrics¹⁷, CCCC¹⁸ and Testwell CMTJava¹⁹.

Finally also tools for providing quality assurance exist which combine all of the above characteristics. Examples of such tools are JTest and C++test by Parasoft²⁰.

Continuous integration and frequent builds

In this section we will discuss technologies which facilitate continuous integration and frequent builds. Software Configuration Management (SCM) systems accomplish this task since they help to manage the evolution of the system. We will describe the two most common ways in which this is achieved by SCM systems. Firstly, version control is used to manage the multiple versions of the system that arise during the evolution of the system; thus making the evolution of the system concrete and controllable. Perry [115] distinguishes three types of versions; *successive* versions which result from the small corrections and improvements, *parallel* versions which result from providing alternate implementations or divergent functionality and *composed* versions which result from constructing a component from separate pieces. Examples of systems that provide version control are SVN and CVS.

Secondly SCM systems can provide functionality to ease the specification and maintenance of the configuration of the system to be built; how the separate components must be combined in order to create the system. To be able to build the system both a description of the structure of the system and information about how to derive object code from the source code modules, are needed. An example of an early tool is Make [54] in which both the description and the information is specified in a resource, the makefile, that was used to derive object code and to link it together. Other examples are Apache Ant²¹, Apache Maven²², Rational Clearcase²³, Telelogic Synergy²⁴, Perforce Jam²⁵ and SCons²⁶.

Which software development related technologies are best equipped to support the agile aspects

The relation between specific benefits and challenges associated with the incorporation of the agile aspects and the software development related supporting technologies is not as clear as with communication related technologies. This is because the challenges and benefits associated with the agile aspects have to do with the *very concept* of both quality

¹⁶<http://pmd.sourceforge.net>

¹⁷<http://metrics.sourceforge.net>

¹⁸<http://sourceforge.net/projects/cccc>

¹⁹<http://www.testwell.fi/cmtjdesc.html>

²⁰<http://www.parasoft.com>

²¹<http://ant.apache.org>

²²<http://maven.apache.org>

²³<http://www-01.ibm.com/software/awdtools/clearcase>

²⁴<http://www.telelogic.com>

²⁵<http://www.perforce.com>

²⁶<http://www.scons.org>

assurance and continuous integration whereas communication is a means to reach many goals. Besides this it is also different from the communication related technologies since both quality assurance and continuous integration are not significantly more difficult because of the distributed nature of the development of software; the requirements that are needed in the tools are not different from those in the co-located case. However to support certain challenges and benefits certain supporting technologies are more significant than others. For example when exploiting *B₇*, *Configuration management is less of an issue* SCM functionality which eases the specification and maintenance of the configuration of the system to be built is more important than a version management tool.

10.3 Integration of collaborative technologies

In the previous section we described technologies which facilitate collaborative software engineering. In this section we will discuss technologies which attempt to combine the functionalities of these technologies. The advantage of integrating the various technologies is that information available in or as a result of one technology is available to another which can be improved by using this information. An example of this is a tool which combines the identity of the last committer of a certain branch in the repository to an instant messenger user. Using this information the integrating technology can make the name of the last committer clickable, opening up a direct instant message connection with him or her using the instant messaging software.

Existing technologies like this have emerged with different initial goals in mind. Firstly integrated development environments (IDE) like Eclipse²⁷ and Microsoft Visual Studio²⁸ have been combined with collaborative technologies such as code-repositories and functional testing, integrating these in the process. Examples of these extensions of IDEs are: Jazz²⁹, Merlin Toolchain³⁰, Team Foundation Server (TFS) and Visual Studio Team System³¹. Secondly there are hosts for open source projects like SourceForge³², Launchpad³³, Google Code³⁴ and Microsoft CodePlex³⁵. These technologies have been extended with all sorts of functionality regarding collaborative development such as: code review, build systems and bug tracking. Thirdly there exist systems created to centralize the building of the system. Examples of such systems are Apache Continuum³⁶, CruiseControl³⁷ and Tinderbox³⁸. These systems, in turn, have also been extended with communication related tech-

²⁷<http://www.eclipse.org>

²⁸<http://msdn.microsoft.com/en-us/vstudio>

²⁹<http://jazz.net>

³⁰<http://merlintoolchain.sourceforge.net>

³¹<http://www.microsoft.com/visualstudio/en-us/products/teamsystem>

³²<http://sourceforge.net>

³³<https://launchpad.net>

³⁴<http://code.google.com>

³⁵<http://www.codeplex.com>

³⁶<http://continuum.apache.org>

³⁷<http://cruisecontrol.sourceforge.net>

³⁸<http://www.mozilla.org/projects/tinderbox>

nologies like scheduling software and quality assurance functionalities such as functional testing and code metrics. Finally there exist technologies which aim to integrate different communication related collaborative technologies. Collaborative virtual environments (CVE), like Croquet³⁹, are examples of this. They mainly aim to combine synchronous forms of communication, such as text, audio and video based communication, however also can be used to share knowledge. For one it can be seen what other project members are doing by checking their status in the virtual environment. Secondly also information regarding the project, like the content of Scrum board can be placed in the virtual environment. A final example of a technology which attempts to integrate different communication related collaborative technologies is a tool such as Zimbra⁴⁰ which integrates an email client, with scheduling software, instant message, task management, wiki and versioning management. In general, technologies which primarily aim for any of these functionalities are being extended in the same fashion. Examples are audio and video based communication in Google Talk⁴¹ and Microsoft MSN and calendar support in Gmail⁴² and Microsoft Outlook⁴³.

Basically, all the aforementioned technologies have similar goals regarding the integration of collaborative technologies. Some technologies which arose from quite different areas of concern, such as hosts for open source projects and centralized build environments, where extended to a degree that it is difficult to differentiate between them. The functionality of all these technologies which primarily concerns us in this thesis is how the different collaborative technologies are best combined, not what the initial goal was to integrate the respective technologies. Therefore we will continue this section by discussing the most interesting integrations of collaborative technologies we found in the types of technology we discussed. Firstly it is possible to integrate tools regarding requirement, testing and project management. The Merlin Toolchain is an example of a tool which performs such an integration of tools for the Eclipse IDE. The added value of this is that information available in one tool is available to improve the functionality of another. An example of this is integrating test and requirement management tools since this integration makes it feasible to display which of the requirements is covered in which of the tests. Likewise, it can be advantageous to link tasks with corresponding bugs and bugs with related requirements and tests.

Secondly, quality assurance, automated build tools and SCM tools can also be integrated to be able to continuously integrate and test the system. Examples of tools which allow for this sort of integration are centralized building systems like the Apache continuum but also TFS, an extension of the Visual Studio IDE. TFS offers a wide range of functionalities also covering the functionalities concerning the integration of requirement, testing and projects management tools discussed above. The combination of quality assurance and automated building is not restricted to functional testing but concerns all sorts of quality assurance. It is for example also possible to continuously measure the code metrics as well, making it

³⁹<http://www.croquetconsortium.org>

⁴⁰<http://www.zimbra.com>

⁴¹<http://www.google.com/talk>

⁴²<http://gmail.com>

⁴³<http://office.microsoft.com/en-us/outlook>

10. HOW TO SUPPORT THE INCORPORATION OF AGILE ASPECTS INTO THE GSD PROCESS WITH TECHNOLOGY

possible to respond immediately when certain metrics approach or reach unacceptable values.

Thirdly, tools which implement continuous integration also provide a means to visualize the results. Usually this is made available in some sort of knowledge base, like a wiki, so it is communicated via asynchronous indirect communication. Apache Continuum, for example, is also configurable to notify certain people via direct communication such as email and instant-messaging and synchronous communication such as IRC.

Another example of an interesting added value of the combination of collaborative technologies arises in performing code reviews. When performing a code review the reviewer needs to have a view of the code and a way to communicate his findings. The added value of integrating the support for this in a single tool is that by combining the code-view and the communication tool, issues reported by the reviewer are more easily linked with the section of source code to which they refer. Also, the information regarding this communication can stay linked with the code and be available for future reference.

Finally, the various communication related technologies can be integrated. In a CVE, it is for instance possible to directly communicate with a colleague using a form of communication supported by the CVE, like text-chat or audio-chat. Besides direct communication however, it is also possible to use the CVE to deduce status information regarding your colleagues, by seeing if their avatar is available, and share information, by for example putting up a big sign in the virtual environment explaining how to perform a specific task. CVEs attempt to integrate the different communication related technologies in such a way, that it most closely resembles same-site development. A tool like Zimbra, on the other hand, does not attempt to reproduce the look and feel of the natural work environment of a collocated team but rather attempts to connect the various communication tools people use, as seamless as possible. Examples of this are the integration of an email client, scheduling software, an instant messenger, task management software, a wiki and a versioning management tool.

Part V

Conclusions and Future Work

Chapter 11

Conclusions and Further research

In this chapter we start by discussing how this research contributes to the existing body of knowledge regarding global software development in general and the agile development approach in a globally distributed environment, in particular. Subsequently we will attempt to answer the research questions by drawing conclusions from the results we gathered. In the third section we will discuss the limitations of our work concerning completeness and ambiguity aspects of our research. Finally, we will conclude by making recommendations concerning areas of interesting research which extends our research.

11.1 Contributions

In this study we researched how the combination of GSD and agile software development is best supported by technology. This is useful because GSD is becoming increasingly interesting these days due to the globalization of business and agile software development allows for flexible development. Because of the flexibility of agile software development and the emphasis on informal communication, it is also a way to alleviate the difficulties associated with GSD. In the existing literature it is suggested that research regarding the incorporation and support of agile software development is warranted. Despite this, however, actual research is still quite scarce and so we chose to contribute to the general body of knowledge by creating a fundament for further research. Next to this we also created a framework to assess how different types of technological support help with respect to certain aspects of agile GSD. We used this framework to discuss what kind of technological support is useful with respect to each challenge and benefit of agile GSD.

We started by discussing GSD in general. To do this we created an overview of all benefits, challenges and solutions to the challenges of GSD in general. In the overview of the challenges, the challenges were grouped by their respective problem areas. Following this we discussed agile software development. This was done by defining what exactly is meant with agile software development by comparing it with heavyweight processes and providing an overview of a number of well-known agile methodologies. Based on this discussion we identified and discussed eight aspects of agile software development and concluded that

five of these aspects are capable of affecting the distances faced in a GSD environment. Subsequently we discussed how incorporating these five aspects into the GSD process:

1. can help to alleviate the consequences of the distances
2. can be problematic because the distances aggravate the difficulty of carrying out the aspects
3. can be improved by using standard solutions from the literature to deal with the challenges meant by point 2.

Finally we switched the focus towards technological support. To do this, first we proposed categories for technological support and checked their applicability by showing they are all beneficial with respect to at least one benefit or challenge of GSD in general. Subsequently we extended the requirement categories with a fifth category in order for them to be applicable with respect to agile GSD as well. Following this we discussed per aspect which benefits and challenges can be supported by which of the defined categories. We concluded by discussing how this support can be achieved by existing technological solutions. The entirety of the aspects, their associated benefits and challenges, and the categories of technological support supporting them, including specific technological solutions, constitutes how technological support can be used to support agile GSD.

11.2 Conclusions

The main question we attempted to answer in this thesis is the following:

"What are the advantages and challenges of the combination of the agile and distributed development approaches and how is technological support best used to deal with these?"

The first part of the question was answered by defining aspects of agile software development and providing an overview of the benefits and challenges these aspects cause in a GSD environment. The second part of the question was answered in threefold. Firstly five categories of technological support for agile GSD were defined to be able to discuss technological support in a structured fashion. Secondly how each of the individual benefits and challenges are best supported by technology was discussed using these requirement categories. It is, however, difficult to draw conclusions regarding the most appropriate technological support for a general project because this depends on the specific project for which it is attempted to derive this. It is for instance possible to not incorporate all aspects of agile software development, in which case part of the table can be neglected. Next to this all challenges and benefits associated with the aspects elected to be incorporated into the project will have varying degrees of importance in different projects. Because of this, the challenges and benefits will have different prioritizations in different projects and drawing conclusions for the general case is difficult.

The third and last aspect of how the second part of the main question was answered regards

the discussion of explicit, existing technological solutions and how these are connected with the requirement categories and the explicit support of benefits and challenges. From this discussion we conclude that the integration of the various collaborative technologies is advantageous since this enables the various technologies to make use of each others data which in turn makes it possible to provide extra functionality and seem like a single system to the end user.

11.3 Reflection

The first limitation of our research concerns that the challenges and benefits are not mutually comparable with respect to their importance. This is because the importance of each of the challenges and benefits is likely to depend on each specific practical setting. For example: a company working within the same time zone suffers less from challenges caused by the temporal distance, and therefore will probably prioritize them differently than a company which is scattered all over the globe. The consequence of this is that it is not possible to derive from our work a strategy to select specific challenges and benefits to try and support, because they are most important in general. However, it is possible to determine how best to support *specific* benefits and challenges.

The second limitation of our research concerns the completeness of all defined benefits, challenges, requirement categories and agile aspects. We cannot guarantee the completeness of these because we have found no method to derive them in a way that is both conclusive and exclusive. In other words; it is possible there are other challenges, benefits, requirement categories and agile aspects we did not elicit. The negative impact of this limitation on our research, however, is limited. For one, the benefits and challenges discussed are derived from an extensive literature research. The challenges and benefits of GSD in general are derived from a particularly large set of resources. With respect to the requirement categories the derivation is based on a much smaller set of resources. The four initial categories closely resemble requirement categories defined by Carmel [29] and to further argue their applicability we showed that each of these categories supported at least a single characteristic of GSD. The same was done for the category which was added to make the categorization applicable for agile GSD as well. Finally the agile aspects cannot be directly founded in literature at all. To correctly found the agile aspects we discussed how they are interrelated with existing methodologies. This was done by discussing how the aspects are reflected in the practices and core values of the Agile Manifesto and eXtreme programming, and how they are reflected in Scrum.

Thirdly, we perform categorizations which posses an inherent subjective characteristic. This applies to both the assignment of agile principles to the agile aspects and the assignment of requirement categories to benefits and challenges. We dealt with this limitation by performing the categorizations in a structured fashion. Another thing that helped deal with this limitation were the two different viewpoints of both authors which arose from the two different research assignments.

Subsequently, aspects, challenges, benefits and requirement categories are not guaranteed to be completely disjunctive. It is attempted to create groups which have the least possible amount of overlap, but still because of the linguistic nature of the concepts, some overlap is unavoidable. For example, having regular and frequent discussions between all members of the development team as part of an iterative approach is part of both aspect A_1 , *Close collaboration among the members of the development team*, and A_2 , *Short iterations, frequent builds and continuous integration*, as this both concerns close collaboration in a development team and following an iterative approach.

Another limitation is concerned with the overview we provide concerning the relationship between aspects, benefits, challenges and distances. Here we focused on how the incorporation of the aspects influenced the distances, either in a positive or negative way, via its associated benefits and challenges. Because of this, we have only shown the influence the aspects have on each other by influencing the distances. This is not the entire story because the aspects also influence each other directly. An example is that aspect A_2 causes the more frequent building of an intermediate system and aspect A_4 , *Customer involvement*, directly benefits from this because it allows the customer to give feedback, on the most recent build, more frequently. The reason we do not discuss these direct interdependencies in this thesis is that our focus lies on the combination of the GSD and agile development approaches and not on the interaction of the various aspects of agile development.

Finally, the overall limitation of this research is that it is limited to a theoretical analysis of available literature and deduction of theoretical aspects, benefits, challenges and requirement categories. These are fully based on literature and deductions from this literature. As such our findings are well-founded in theory and traceable to its sources. However, our research results have not yet been validated as a whole. As such, we will continue with the validation of these findings by means of a series of industrial case studies. These are expected to complete and/or confirm (parts of) our findings and extend the work with guidelines on how to exploit them in practice.

11.4 Recommendations for further research

Having performed the research described in this thesis several interesting possibilities of further research regarding agile GSD have emerged. Firstly to further objectify the various categorizations we performed as well as validate the completeness and disjunctiveness of the various benefits, challenges and aspects we defined, an expert committee can be used. Secondly the mutual importance of benefits and challenges can be researched. On the one hand this can be done for a specific project, concluding with an advice regarding the kind of technological support that is most beneficial for that particular project. On the other hand, various different projects can be researched, in order to deduce the dimensions of a project which particularly influence the mutual importance of the benefits and challenges. In this fashion several categories of projects with similar requirements with respect to the tech-

nological support can be defined. Thirdly, the research regarding the mutual relationships between aspects, distances, benefits and challenges can be extended by explicitly researching the mutual relationships between aspects. Fourthly, the theoretical conclusions we drew in this thesis should be evaluated in practical cases. This not only regards the applicability of the framework but also, for example, how well the non-technological solutions to the challenges faced when incorporating agile aspects into GSD, work. Finally it is interesting to research how the overall integration of collaborative technologies supporting agile GSD can be improved upon. In such a research it can be researched which technologies profit most from being integrated and what view of such an integrated system are most valuable for its users. Examples of such views are a code-centric view, most closely resembling a classic IDE, and a communication centric view, attempting to mimic a normal, co-located, work environment. Such a research can be quite large; therefore we will mention some examples of smaller research projects concerning this research. Firstly, to integrate communication related technologies, researching what communication lines are used most often in projects is valuable. Secondly researching the applicability of existing, but not yet widespread, communication technologies, in the context of project development can be quite useful as well. Examples of such technologies are virtual meeting software and Collaborative Virtual Environments (CVE). Lastly, researching specific types of project collaboration requiring quite extensive communication, like pair programming and code review, can be useful to elicit the requirements of an environment integrating all technologies necessary to support agile GSD sufficiently. This is plausible, since if the most communication demanding activities can be sufficiently supported by such an environment it is likely that this is true for most other activities as well. We propose to call such a development environment an Integrated Collaborative Development Environment (ICDE).

Part VI

Practical Research

Chapter 12

Practical Work

In the previous chapter we concluded that there is a lot to gain with respect to supporting agile GSD with technology by integrating the supporting technologies into an Integrated Collaborative Development Environment (ICDE). In this chapter we attempt to make a first step into constructing such an ICDE. This chapter is structured as follows: Firstly we describe the research design we have adopted. Subsequently we discuss the context in which we did the practical work. Following this we discuss how the research described in the research design was performed and what the results we acquired. Finally we discuss the validity of the practical work described in this chapter and recommend further research.

12.1 Research design

The goal of the practical part of this master thesis is to make a first step into the construction of an Integrated Collaborative Development Environment (ICDE). Therefore our initial research question is:

”What are the requirements of an Integrated Collaborative Development Environment to support agile GSD?”

In order to elicit these requirements we conduct qualitative research. With the term qualitative research a type of research is meant that produces findings not arrived at by statistical procedures or other means of quantification [45]. We primarily choose qualitative research as opposed to quantitative research because of the emphasis on interpersonal relations in the research problem. Qualitative methods are particularly appropriate to study the complexities of human behavior, for example regarding communication and difficulties in understanding [136]. Next to this, qualitative studies can generate well-grounded hypotheses and findings that incorporate the complexity of the phenomenon under study [121] which is precisely what we attempt to accomplish in this research. This incorporation of the complexity of the phenomenon under study originates from the qualitative approach since such methods force the researcher to delve into the complexity of the problem, rather than abstract it away [107, 142]. The alternative to qualitative research, quantitative research, is less appropriate

in this particular setting, since much of human behavior cannot be adequately described and explained through statistics and other quantitative methods [136].

Qualitative research attempts to develop an understanding or interpretation that answers the basic question of what is going on. This is done through an iterative process that starts by developing an initial understanding of the setting and perspectives of the people being studied. This understanding is then tested and modified through cycles of additional data acquisition and analysis until an adequately coherent interpretation is reached [102, 112, 88]. We start this research with preliminary knowledge regarding existing supporting technologies and common problems and solutions in the field of agile GSD. This knowledge originates from our research discussed in the previous parts of this thesis. It should be extended with practical insights, to be able to construct a categorized list of requirements of an ICDE. This is true, since the practical validity of the problems and solutions need to be tested and because the knowledge can be extended with new insights to create a more complete list of requirements. The method we choose to acquire these practical insights, is interviewing people in four different companies performing the roles of manager, architect and developer in projects of an agile and globally distributed nature. We choose the method of interviewing rather than observation or participant-observation because we expect interviewing to be a very direct and time-efficient method for extracting practical insights from experts. Disadvantages of interviews are that they are subject to common problems such as bias, poor recall, and poor or inaccurate articulation [143].

Interviews can be divided into three categories based on the degree of structuring: structured, semi-structured and unstructured interviews [55]. The goal of structured interviews is to capture precise data of a codable nature so as to explain behavior within pre-established categories and consist of pre-established questions with a limited set of response categories and generally little room for variation [55]. The goal of unstructured interviews is to understand the complex behavior of members of society without imposing any a priori categorization that may limit the field of inquiry [55, 122]. In our research we choose the third type of interviews: semi-structured interviews. This kind of interviews possesses qualities of both structured and unstructured interviews. In the interviews we perform, we do have a list of general questions but we also deviate from this list, make up extra questions as we go along and allow the interviewees to bring up issues of their accord. We choose this type of interviewing because we feel this leaves the most room for well informed informants to provide us with important insights. Disadvantages of interviewing in this fashion are concerned with the fact that data obtained from interviews are likely to be biased due to the interviewer influencing the interviewee with his own preconceived ideas [126]. Additionally, because of the semi-structured nature, the length of such interviews is harder to estimate and control, and because of this the chance of running out of time is more profound.

Following the acquisition of data we analyze the data. Qualitative analysis is not the quantification of qualitative data but rather the nonmathematical process of interpretation, carried out for the purpose of discovering concepts and relationships in raw data and organizing these into a theoretical explanatory scheme [45]. The main technique we use to analyze the

data gathered using the semi-structured interviews discussed above, is coding because we attempt to create a categorized list of requirements. In qualitative research the purpose of coding is not, applying a pre-established set of categories to the data according to explicit, unambiguous rules, with the primary goal being to generate frequency counts of the items in each category, like in experimental or survey research or content analysis [88]. Instead it involves selecting particular segments of data and sorting these into categories that facilitate insight, comparison, and the development of theory [45]. The coding categories that are defined during the analysis may be deducted from the evaluation questions, existing theory, or prior knowledge of the setting and system but they may also be developed inductively by the evaluator during the analysis or taken from the language and conceptual structure used by the people studied [88]. Next to coding we also create analytical memos to be able to convert our perceptions and thoughts into a visible form that allows reflection and further manipulation [45, 102].

The analysis of the data gathered from the semi-structured interviews resulted in a categorized list of requirements of an ICDE which were subsequently validated by the interviewees. These requirements mainly concern the integration of several existing forms of technological support. Using this list of requirements and again the data gathered from the semi-structured interviews we deducted a number of ideas, which integrate certain existing technologies, to try out in a feasibility study. Following this we validated and improved upon these ideas by follow-up interviews with people from the same companies as the initial semi-structured interviews. This resulted in a technical feasibility study which was conducted subsequently in cooperation with the companies. This feasibility study was conducted to show the technical feasibility of a subset of the ideas we defined.

12.2 Context

During our entire research project we were involved in a knowledge group set up around the research project. This group consisted of the following companies: Exact Software, IHomer, Mavim, SDL Tridion and Xebia. The primary goal of the knowledge group was to exchange knowledge regarding distributed and agile development with the emphasis on the Scrum methodology. The group arranged periodic meetings. In these meetings we discussed our progress in the research project, how each company dealt with distributed agile development and discussed a number of specific themes in the context of distributed agile development. These companies fulfilled different roles with respect to the practical part of our research. Firstly, as mentioned, all companies took part in the meetings of the knowledge group and thus provided us with valuable feedback with respect to the general progress of our research. Secondly, the semi-structured interviews were conducted at Exact Software, IHomer, SDL Tridion and Xebia. Thirdly, the definition of the concepts and brainstorming on ideas for a feasibility study, were done in cooperation with Exact Software, IHomer, and SDL Tridion. Finally, the feedback on the implemented system created during our feasibility study was given by this same group. In the rest of this section we will briefly describe the five companies in the knowledge group.

Exact Software

Institutional Context

Exact Software is established in 1984 and is one of the world's leading providers of business software solutions. The integrated software solutions Exact Software offers include: Enterprise Resource Planning (ERP), Human Resource Management (HRM), Customer Relationship Management (CRM), Corporate Performance Management (CPM), project management and electronic workflow management. Exact Software has offices in more than 40 countries in Europe; the Middle East; North, Central and South America; Asia; Australia and Africa. Currently, there are over 2.500 employees employed by Exact Software worldwide.

Customers

Exact Software provides solutions to small and medium-sized businesses which all have different implementation and industry requirements ranging from manufacturing, distribution and retail to trade and service environments.

Technologies and Methodologies

Exact Software offers a large variety of software solutions, for each of these solutions the most appropriate techniques and methodologies are used.

Distributed Context

Exact Software has structured its global network into four regions: Asia Pacific; Europe, the Middle East and Africa; the Americas and the Netherlands to be able to efficiently accommodate the varied needs worldwide.

Structure

The main activities with respect to corporate R&D level are carried out in Delft and Kuala Lumpur. Each of the four regions mentioned above provide input to R&D by defining business cases and requirements. They each serve their own distinct market, in a different way, but all of these project teams share commitment to achieve a single solution capable of serving all markets. To achieve this commitment, the corporate strategy of Exact Software is developed and defined by the corporate headquarter, located in Delft. Regional development mainly concerns the development of local and custom solutions.

IHomer

Institutional Context

IHomer was founded in 2008 and focuses on software development and maintenance. This organization is completely distributed and has no main office; instead all the employees work at home. IHomer currently employs about 10 experienced employees.

Customers

IHomer is still in its construction phase and has several ongoing projects by different organizations. They try to establish a basis for long term relationships.

Technologies and Methodologies

IHomer applies the Scrum Agile Methodology and uses modern development tools based on the Microsoft platform and Java in order to keep projects manageable and to achieve short turnaround times.

Mavim**Institutional Context**

Mavim was founded in 1990 and has offices in the Netherlands, Australia and New Zealand. It received international publicity through its first product SIS. Because of its international success SIS was later renamed to Rules. Currently, about 250000 users use Rules to structure and optimize their business processes. Besides software Mavim also offers technical and functional support. Examples are trainings, functional and business consultancy and support. Mavim currently has about 60 employees.

Customers

Rules, Mavim his main product, is suitable for companies of any size, whether they are national or international. Therefore, customers arise from various branches. Examples are: construction, financial services, health care, trade, ICT, logistics, government and building cooperations.

Technologies and Methodologies

Mavim is a Microsoft Gold partner and the first company which has achieved Visio Data Visualization Specialist certification in the Netherlands¹. Mavim applies the Scrum methodology for maintenance on and adding features to its existing software. The Scrum methodology is used together with user stories for the development of new products. Mavim uses C#.NET in combination with Microsoft Visual Studio Team System 2008.

Distributed Context

Most developers at Mavim frequently work from home. In the past, Mavim did a distributed project in India who created a satellite application for them.

Structure

Mavim uses Team Foundation Server as main code repository. People working from home can be contacted by Windows Live Messenger, by mail and by company telephone. Working at home is done at least once a week. People work individually both at home and at work, this is caused by a high level of specialization for the maintenance team, but working together is done when necessary. New products and user stories are developed co-located on the main location in Noordwijk.

¹http://visiotoolbox.com/de/articles/News_Announcements/Mavim\%20first\%20Microsoft\%20Data\%20Visualization\%20Specialist\%20in\%20the\%20Netherlands_201.aspx

SDL Tridion

Institutional Context

Tridion is established in 1999 and has focused on enabling customers to manage their content within a web environment. Since 2000, Tridion expanded to other European countries and has opened offices in America and in Asia Pacific. In 2007 Tridion has been acquired by SDL plc and the name of the company was altered to SDL Tridion. SDL Tridion employs around 180 people divided over all sites.

Customers

SDL Tridion provides content management solutions for a wide variety of industries, for example: manufacturing, financial services, travel and tourism, and the public sector. They serve a large variety of businesses, including small sized businesses and multinationals.

Technologies and Methodologies

SDL Tridion applies a methodology which consists of five phases: the prepare stage, the structure stage, the design stage, the build stage and the deploy stage. This methodology is applied to ensure effective and fast Content Management System implementations. SDL Tridion uses both .NET and Java technology to develop their Web Content Management solution.

Distributed Context

Since 2000, Tridion started with distributed projects in which the different teams all performed a modular task.

Structure

SDL Tridion has development centers in Amsterdam, Kiev (Ukraine) and San Jose (United States). Projects are defined in either two ways:

1. Project teams are formed across country borders and tasks are performed by all team members in cooperation. In these projects the daily communication and coordination is essential. These projects have developers and testers in multiple locations working on the same code base.
2. Project teams are formed locally to execute specific tasks. Often a work package is executed by the team and the outcome is shared with the other development centers. Examples in this category: projects which are developed in country A and tested in country B.

Xebia

Institutional Context

Xebia was founded in the Netherlands in 2001, performs large scale development projects, consultancy in architecture and auditing, and helps organizations to manage their corporate IT infrastructure. In 2004, Xebia opened a specialized development center in India and an

independent consultancy branch in France which focuses on architecture and agile consulting. Xebia is an international IT consultancy and project organization, with currently about 150 employees.

Customers

Most of Xebia's consultancy clients are Top 300 organizations. However, they also work with smaller and/or specialized organizations concerning agile transformation and software projects.

Technologies and Methodologies

Xebia focuses on Enterprise Java technology, Agile development methods and outsourcing services.

Distributed Context

In 2006, Xebia started distributed projects with teams made up half of Dutch and half of Indian team members.

Structure

Xebia implemented a distributed software development team model on multiple projects of variable types with teams located half in the Netherlands and half in India. These fully distributed teams use the Scrum process combined with engineering practices taken from eXtreme Programming.

12.3 Findings

In this section we will describe the research we performed and the results we obtained. We will follow the structure described in the section research design and therefore will start with describing the data acquisition associated with eliciting the list of requirements of an ICDE.

12.3.1 Deducing the requirements of an ICDE: data acquisition

We started with preliminary knowledge regarding existing supporting technologies and common problems and solutions in the field of agile GSD, originating from our research discussed in the previous parts of this thesis. To be able to transform this knowledge into a list of requirements of an ICDE we sought to both test and extend it with practical insights. To do so we arranged, with four of the five companies (Exact Software, iHomer, SDL Tridion and Xebia) in the knowledge group we are part of, to conduct several two hour interviews at each of these companies. Because we wanted to determine requirements for an integrated development environment aiming to support working collaboratively in a distributed setting and using an agile paradigm we requested to interview people who most closely resembled the following profile:

- Five years of experience working in IT

- Two years of experience working in a distributed setting
- Two years of experience in his or her current role
- Level of education HBO and above

Because we wanted to get an overall view of how to support a distributed project we also requested to interview people performing different roles in the development process, namely:

- An architect/designer
- A developer
- A product/project manager

Most our requests were honored and we were able to interview each of the different roles at Exact Software, iHomer and SDL Tridion whereas we were able to interview an architect and a developer at Xebia.

In the time span of a week we visited all the companies and conducted the interviews. As discussed earlier these interviews were of a semi-structured nature. We used a scheme of basic questions to guide us through the interviews, elaborating when necessary. This scheme is included in appendix A. It consists of four mayor lines of questioning. The first line of questioning attempts to elicit information regarding the work context of the interviewee to be able to put the information we gather during the interview in the proper context. We planned approximately fifteen minutes for this. In the second line of questioning it is attempted to elicit common user stories and scenarios with respect to collaborative development and the difficulties that arise in a distributed context. To do this we created a list of collaborative activities in the development process and asked similar questions regarding each of these. For this line of questioning we planned sixty minutes because we feel finding out how all collaborative development activities are carried out and what challenges are faced when performing them is an integral part of the goal of these interviews. To show that it is likely we covered all collaborative activities in the development process we explicitly connected the activities we listed with the associated SWEBOK Knowledge Areas (KA) [2] in table 12.1. In this mapping of collaborative activities to Knowledge Areas from SWEBOK we did not cover all Knowledge areas. The ones that were not covered are the following:

1. Software engineering process
2. Software engineering tools and methods

The software engineering process and Software engineering tools and methods were left out because the subjects covered in these Knowledge Areas either fell outside the scope of a project or could not be regarded collaborative activities. The software engineering process knowledge area mainly concerns the selection of a process and software engineering tools and methods mainly concerns supporting methods and tools for the software engineering process. Finally we also cover information regarding the software engineering process in the first and fourth line of questioning (discussed later in this section), respectively about

Activity	Knowledge Areas	Explanation
Planning	Software Engineering Management	This KA consists of more than just the planning activity but this concerns non-collaborative activities and activities that are more closely associated with different knowledge areas.
Requirement clarification	Software Requirements	
Design	Software Design	
Construction	Software Construction	
Quality assurance	Software Testing Software Quality	Here we grouped two KAs because they both concern quality assurance. In the interviews we split this activity into testing and code review and evaluation.
Integration	Software Configuration Management	
Maintenance	Software Maintenance	

Table 12.1: Mapping of development activities with their associated Knowledge Areas

the context and technological support. In the third line of questioning we attempt to elicit the negative impact of working in a distributed setting on informal, non work-related, communication. We feel it is important to attempt to make this explicit since informal, non work-related, communication is a form of social interaction often lost when working in a distributed setting. Because of this it might be the cause of some of the problems and so information regarding this could lead to ways to deal with these problems. We also explicitly asked the interviewees whether they had solutions of their own to deal with these issues. We planned thirty minutes for this line of questioning. The fourth and final line of questioning regards eliciting information regarding the integration of existing technological support. In this line of questioning we split the discussion in three categories:

1. Integration between communication related technologies
2. Integration between software development related technologies
3. Integration between communication related and software development related technologies

For each of these categories we attempted to elicit the current types of integration that are in place as well as ideas to either improve the existing integrations or ideas regarding entire new forms of integration. We planned fifteen minutes for this line of questioning.

Both of the interviewers took part in interviewing each interviewee. Both the researchers

took part in the discussion to allow for a natural flow in the conversation and allow all necessary subjects to come up. We did, however, assign roles to ourselves. One of us would be in charge of following the structure of the interview while the other would make sure sufficient notes were created during the interview for analysis. These notes were created using the same structure as was used for the interviews attempting to write down as much useful information as possible. Between interviews we switched between the two roles mentioned above, both of the interviewers performing each role roughly the same number of times. Finally we asked all interviewees to validate our findings after we analyzed the data and created a categorized list of requirements of an ICDE. How we constructed that list is discussed in the next section.

12.3.2 Deducing the requirements of an ICDE: data analysis

Following the interviews we used coding and analytical memos to use the data to construct a categorized list of requirements of an ICDE. This list answers our initial research question:

"What are the requirements of an Integrated Collaborative Development Environment to support agile GSD?"

Firstly we started to construct an uncategorized list of requirements. The entries of this list were inductively developed from prior knowledge combined with the data gathered from the interviews. Following this we split the requirements into categories based on the explicit goal the requirements attempt to accomplish. The categories, or sub-requirements of an ICDE, we defined are the following:

RC_1 Incorporate the knowledge about the project into the development environment

RC_2 Facilitate inter-personal contact

RC_3 Derive information from the system

We also defined a mapping of these categories of requirements of an ICDE to the integration of requirement categories of technological support introduced in chapter 8. To recap, these categories are the following:

R_1 **Facilitate direct contact between colleagues**

Technological support which facilitates direct communication between two or more actors.

R_2 **Facilitate knowledge sharing among colleagues**

Technological support which facilitates the sharing of technical project knowledge.

R_3 **Facilitate transparency of the project status**

Technological support which facilitates the sharing of organizational project knowledge.

R₄ Facilitate quality assurance

Technological support which facilitates quality assurance functions to monitor and guarantee the quality of the product.

R₅ Facilitate continuous integration and frequent builds

Technological support which eases the process of continuously integrating the system as well as producing builds frequently.

This mapping regards which types of technological support should be integrated into the development environment to reach the requirements of an ICDE of the categories mentioned above. Firstly to achieve the requirements from category *RC₁*, technology from category *R₂* and *R₃* should be integrated into the development environment. This is because technology from categories *R₂* and *R₃* are the supporting technologies with information about the project. Secondly to achieve the requirements from category *RC₂*, technology from category *R₁* and *R₃* should be integrated into the development environment. This is because, in order to facilitate interpersonal contact, technologies which support communication (*R₁*) should be available in the development environment as well as information regarding the status the other people in the project (*R₃*). Finally to achieve the requirements from category *RC₃*, technology from category *R₄* and *R₅* should be integrated into the development environment. This is because technology from categories *R₄* and *R₅* are the supporting technologies from which information about the system can be derived.

Following the definition of these three sub-requirements of an ICDE we further extended and refined the list of requirements and assigned them to these categories. The entire categorized list of requirements is presented in appendix B. The entries within each of the categories are approaches to achieve (parts of) the sub-requirements. Next, we will illustrate the three categories by presenting examples extracted from the entire list of requirements in appendix B.

RC₁ Incorporate the knowledge about the project into the development environment

Firstly this can be achieved by creating connections in the context. An example of this is connecting work items, such as an issue, with information about related actors, like the reporter of the issue and the actor assigned to resolve the issue. Secondly, this can be achieved by means of a 'dashboard': An overview of the current status of the entire project. Examples of such status information are information regarding the quality of the system and information regarding the progress of implementation. Finally, this sub-requirement can be achieved by notifying certain actors when specific events occur. An example is to automatically send an email to the actor that made a commit which broke the system.

RC₂ Facilitate inter-personal contact

Firstly, this can be achieved by providing for different media to communicate, like text audio and video, and facilitating switching between these media. Secondly, it can be achieved by offering clear, accurate and up-to-date status information about relevant actors. Thirdly, this sub-requirement can also be achieved by non-technological

approaches. Examples of this are providing a uniform discussion environment and the exchange of ambassadors between sites to create a stronger connection between these sites.

RC₃ Derive information from the system

A way to achieve this is integrating the functionality of existing systems supporting the creation of the system. An example of this is connecting the continuous build server and the solutions used to provide quality assurance to achieve continuous quality assurance.

Finally, as mentioned earlier, the list of requirements was validated by the people we interviewed.

12.3.3 Ideas for further research

After having made a categorized list of requirements of an ICDE we used this list in combination with the data originally gathered from the interviews to create a list of concepts which warrant further research. These concepts could be actual ideas but also more general propositions for further research. The first of these concepts we defined is to research the usage of a "*mandatory*" open line, between virtual neighbors (people that are each others specific remote contact person), on certain pre-defined times. This idea is related to the virtual neighbor (or buddy) system discussed in the list of requirements. By opening a communication line between two virtual neighbors, for example once a week for thirty minutes, they are able to keep in touch and talk about unplanned subjects. The idea is to mimic the interactions they have with the person physically sitting next to them. Research regarding this idea would concern gathering the requirements for such a scheduled conversation with respect to for example duration and frequency; deducing how this could be best supported by technological support and ultimately trying it out in an actual project setting, followed by an evaluation.

The second concept we defined is to research the concept of an '*ongoing conversations list*'. The basic idea with respect to this concept is to mimic the overhearing of conversations in ones local workspace. In such a setting, people learn things from overhearing conversations and are also able to jump in and contribute. Basically an '*ongoing conversations list*' would entail the following:

1. List of the current ongoing conversations (via chat, audio, video etc.) with the participants and a topic, question or issue.
2. This list could be shown on a central overview screen.
3. Basic functionality:
 - a) People can start an "*open conversation*" either with explicit other participants or without.

- b) People can join a "*open conversations*" to be able to read/see/hear it (either as a observant or participant) and it is clear to everyone who is monitoring the conversation.
 - c) It is possible to switch between the different conversation media.
- 4. It should be possible to switch from normal (*private*) forms of communication to an open conversation (and vice versa)

Other names that describe a list described here well are '*topic pool*' and '*synchronous forum*'. This research could entail the construction of a prototype, testing it out in a project setting and evaluating the progress.

The third concept we defined has to do with the creation of a single overview of all relevant project related data. Such an overview is also known as a '*dashboard*'. There exist solutions which provide an overview of project related data but we feel these solutions could be extended. Research regarding this concept could entail both *what* project data is relevant to show and *how* to acquire the necessary data. Examples of project information which could be shown on a dashboard are the following:

1. Functional/Integration/regression/metrics test status
2. Organizational information
 - a) Indicator for percentage of planned tasks which are completed
 - b) Divergence with respect to planning
 - c) SCRUM board / burn down chart
3. Overview of configuration machines (both for testing and maintenance) and if they are patched sufficiently

The fourth concept we defined concerns researching: what information regarding the current activity (status) of a colleague is beneficial, which of this information can be gathered and how it helps both with respect to collaboration and making the development team feel more like a team. Next to this it could also be researched how best to integrate the status information into the environment of the developer. Examples of ways to extend the status information of remote colleagues visible to members of the development team are the following:

1. Including the application people are currently using in the status (for example the status used in OCS).
2. Pointing an always on global-camera on the work space of all developers. This way it is possible to see what colleagues are available at their work-stations and how busy they are (which can be deducted by looking at them).
3. Adding the current view of the developer in the status. Examples of this are:

- a) Looking at bug #xxx in the bug tracker
- b) Currently working on task #xxx
- c) Documenting on subject Y in the wiki

The fifth and final concept we defined is quite general in nature. It regards researching how to lower the threshold for using the various communication media which are available. This regards exploring the current types of communication available and how (and when) they are being applied, but also when (and why) they are not being applied when they could be beneficial. We feel this is important as we gathered from the interviews that quite some means to communicate that companies have in place are not used or could be used more. This research could lead to concrete advice regarding how to make better use of communication media currently in place, the quality of these media and what communication media could prove to be beneficial when added.

12.4 Feasibility study

With the five concepts we defined in the previous section we returned to the companies to determine what research we could best perform in the context of our research. After having a number of discussions we decided to integrate three of these concepts in one single system after this called 'our system'. Firstly, our system will provide for an '*ongoing conversation list*' with the possibility to join the ongoing conversations. Secondly, our system will provide an overview of the status information of the members of the development team. And thirdly, our system will provide an overview of relevant project information. The information necessary to create such a system should come from the technological support the development team uses. For this feasibility study we elected two such systems which are often used in practice: Microsoft office communications server 2007² and Team Foundation Server 2008³ and their associated clients: Microsoft office communicator 2007⁴ and Visual Studio Team System 2008 Team Suite⁵ respectively. We chose to use these two specific systems because most companies from the knowledge group use these and they contain the information necessary to implement the three concepts.

Office Communicator (OC) is an instant messaging program, used with Office Communications Server (OCS), aimed at corporate environments. It provides features like instant messaging; Voice over IP, Video conferencing and Screen Sharing. We use this system to gather information regarding the ongoing conversations and the status of the members of the development team. Next to that we add the possibility to request to join an ongoing conversation. Visual Studio Team System (VSTS) and Team Foundation Server (TFS) together make up a set of software development, collaboration, metrics, and reporting tools. In this TFS takes care of the data storage and collaboration backend and VSTS takes care of the

²<http://www.microsoft.com/communicationsserver/en-us/default.aspx>

³<http://msdn.microsoft.com/en-us/teamssystem/default.aspx>

⁴<http://office.microsoft.com/en-us/communicator/HA102037151033.aspx>

⁵<http://www.microsoft.com/products/info/product.aspx?pcid=ef08ccd2-8cd8-44ae-81a4-f7bc63f007d4>

visualization of the data to the developer in an integrated development environment. TFS provides a source control repository, work item tracking, reporting services, build services and project management capabilities. We use this system to gather information regarding project related items like sprint backlog items and also combine this data with user information to extend the view of the status of the members of the development team (for example what sprint backlog items a specific user is working on). Finally, we extract overviews from this system and incorporate these in our system.

In the rest of this section we will start by describing the general context of the feasibility study. To do this, we will first describe the functionality of both source systems in more detail. Following this we will describe what exactly we are going to display in our system and explain the functionality we added. Following the context we will discuss our technical implementation. Subsequently we will discuss how to use the user interface of our system. We will conclude by validating this feasibility study by discussing the feedback we acquired about our system from the three companies that helped define this feasibility study.

12.4.1 Context

Office communications server⁶

Microsoft Office Communications Server is an enterprise real-time communications server. It provides the infrastructure for enterprise instant messaging, presence information, file transfer, peer to peer and multiparty voice and video calling, ad hoc and structured conferences (audio, video and web), connectivity with the general telephone network (Public Switched Telephone Network or PSTN) and desktop and application sharing. These features are available within an organization, between organizations, and with external users on the public internet or standard phones on the PSTN. Microsoft Office Communicator 2007 and the LiveMeeting console (LMC) 2007 are the primary client applications released for OCS. OC 2007 is the client used for IM, presence, voice and video calls and ad hoc conferences. LMC is used for more structured meetings, conferences and application sharing. It can run natively against either OCS or the LiveMeeting hosted service. In this feasibility study we regard the use of Office Communicator and we will discuss its basic usage. First we will discuss the use of presence information in Office Communicator. Following this we will discuss the functionality of Office Communicator regarding communication between contacts. We will conclude the discussion of Office Communicator by explaining which portion of the information we chose to incorporate in our system and how we extended the communication related functionality.

*Presence information*⁷

A person's presence is determined by a collection of attributes that describe the person's

⁶Based on http://en.wikipedia.org/wiki/Microsoft_Office_Communications_Server

⁷Adaptation of <http://office.microsoft.com/en-us/help/HA102067221033.aspx>

status, activity, location, willingness to communicate, and contact information. Presence information helps you to contact others and helps others to reach you. In figure 12.1 the dimensions of status available in Office Communicator are displayed and in figure 12.2 it is shown how this information is displayed in its user interface.



Figure 12.1: Presence dimensions

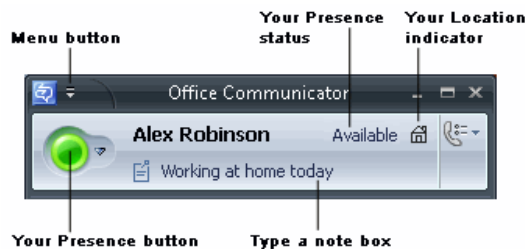







Figure 12.2: Presence in Office Communicator

Presence information provides context for a contact and helps you to decide the best way to communicate with the contact. For example, assume that you need to discuss a proposal with a co-worker. You look at his status in your Contact List and see that the contact is available. You could walk down the hall and talk to the person face-to-face, but you notice the person's location indicator and personal note indicates the person is working at home, so you decide to send an instant message to the contact instead. Communicator displays various bits of information about each contact, so you can see the contact's context - are they available; in a meeting; free in an hour; working at home - and you can use that context to make a decision about the best way to communicate with the contact. For example, if the contact is in a meeting and you have urgent business to discuss, you can send him an instant message. Figure 12.3 shows how the status of your contact is shown in the communicator interface and also a number of options you can use to contact them.

As can be seen in figure 12.2 each contact has a presence button that reflects the contact's status. A contact's presence button changes color based on a variety of factors. This is described in table 12.2.

Presence button	Status text	Description
	Available	The contact is online and can participate in conversations. This status can be set

Presence button	Status text	Description
		manually by the user.
	Busy In a Call In a Conference In a Meeting	<p>The contact is available but engaged in another activity. Activities include:</p> <ul style="list-style-type: none"> • In a call The contact is in a phone, voice, or video conversation. • In a Conference The contact is in a multiparty conversation using phone, voice, or video. • In a Meeting The Office Outlook calendar shows the contact has a scheduled meeting. <p>This presence level can be set manually by the user.</p>
	Do not disturb	<p>You see this status for a contact if the contact has assigned you to an access level other than the Team access level and the following condition exists:</p> <ul style="list-style-type: none"> • The contact has manually set his or her presence status to Do Not Disturb.
	Urgent interruptions only	<p>You see this status for a contact if the contact has assigned you to the Team access level and the following condition exists:</p> <ul style="list-style-type: none"> • The contact has manually set his or her presence status to Do Not Disturb.
	Away Out of office	<p>The contact is probably not available. This status is displayed for the following reasons:</p> <ul style="list-style-type: none"> • The contacts computer has been idle for more than the idle time period setting 15 minutes by default. • The contacts Office Outlook calendar or Out of Office Assistant indicates that he or she is out of the office. • The contact is temporarily unavailable. As soon as activity is detected on the contacts computer, Communicator 2007 automatically resets the presence status to the appropriate state. • The contact has manually set his or her presence status






Presence button	Status text	Description
		to Away.
	Inactive	This contact may be available, but his or her computer has been idle for more than the idle time period setting five minutes by default. In this state, the contact is online and transitioning from an Available state, as indicated by the half-green/half-yellow button. This status is set by Communicator.
	Busy (Inactive)	This contact is engaged in a meeting or is scheduled to be in a meeting (as indicated in the Outlook calendar) , but his or her computer has been inactive for the idle time period setting 5 minutes by default. This status is set by Communicator.
	Offline	<p>The contact is not available. This status is displayed for the following reasons:</p> <ul style="list-style-type: none">• The contact has manually set his or her presence status to Appear Offline. (Appear Offline is not available by default. The system administrator must enable it for an organization using the group policy: EnableAppearOffline.)• Communicator is not running on the contacts computer, or the contact has not signed-in.• The contact has blocked you from seeing his or her presence status.
	Unknown status	Communicator cannot determine the status of the contact. This status is usually displayed because the contacts presence status is stored in another computer system, such as that of an organization that is not a federated partner.
	Blocked	This button is displayed in your Contact List next to the contact name you have blocked. To the person you have blocked, you appear to be offline.

Table 12.2: The different states of the presence button

Communication functionality

In figure 12.4 the drop-down menu displayed when right clicking one of your contacts in the Communicator user interface can be seen. This shows the different options available to communicate with a colleague using Communicator:

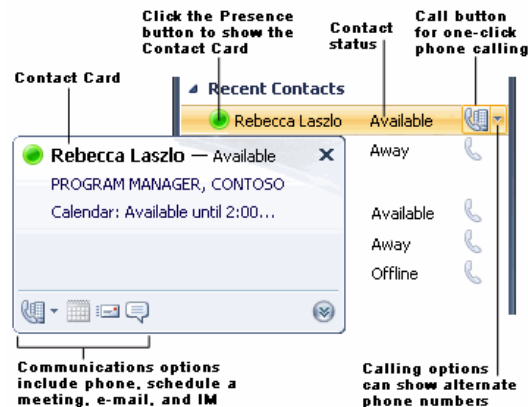


Figure 12.3: Presence information of one of your contacts

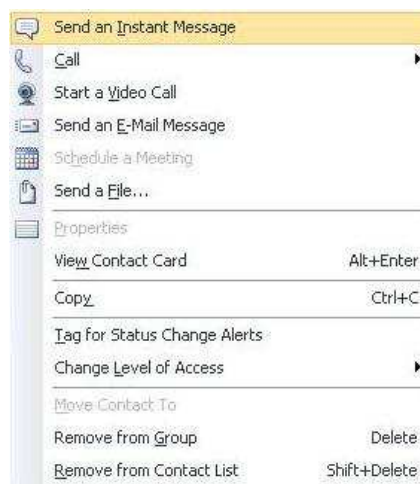


Figure 12.4: Available communication options in Office Communicator

1. *Send an instant message.* Text based
2. *Call.* Audio based
3. *Start a video call.* Video based
4. *Send an email message.* Text based, asynchronous

Besides these options to communicate Communicator offers more ways to enrich the communication. When in an instant message conversations with someone (having clicked *send an instant message* in the previously mentioned drop down menu) a window such as shown in figure 12.5 can be seen.

From this figure it can be gathered that from within a conversation it is possible to do a number of things. Firstly it is possible to change the type of communication between text based,



Figure 12.5: The conversation window in Office Communicator

audio based and video based. Secondly it is possible to send a file to a contact. Lastly it is possible to invite more people into the conversation, turning the conversation into a multi participant conversation.

Office Communicator in connection with this feasibility study

In this feasibility study we use Office Communicator to gather information to incorporate the user status in our system and we extend the functionality to create an open conversation list. To incorporate the user status we gather various types of information from communicator. Firstly the information about a user:

1. Real name
2. Email address
3. User note
4. Communicator status
5. Out Of Office status
6. The time the user has been away

The decision to collect these types of information is made intuitively based on the interviews and literature findings. We do feel there are grounds to assume however, the specific information that will be beneficial will depend on the specific context in which the solution will be used. Secondly we gather information regarding conversations. We show for each

user which conversations they are part of or have been part of. About each conversation we record the following:

1. Start time
2. Participants
3. History
4. Active status

Besides displaying information, regarding current and past conversations with Office Communicator between members of the development team, we also extend the functionality of Communicator. In Office Communicator only people in a specific conversation have the ability to add other people to that conversation, or, for that matter, know the conversation is going on. When working co-located you often notice when colleagues are having a (verbal) conversation. When you are interested in the content of the conversation you can listen-in on the conversation or even join the conversation. Another possibility is your colleagues are trying to resolve an issue you know the answer to and you can help them resolve it. We are looking to extend the functionality of Office Communicator to more resemble the co-located situation. For that reason we show the list of ongoing conversations and offer the possibility of asking to join them. When a member of the development team requests to join a conversation, the leader of that conversation is given the choice whether or not to allow this. Finally, we also show a list of past conversations to allow members of the development team access to potentially valuable project knowledge contained in the past conversation data.

Team Foundation Server⁸

Team Foundation server provides source control, data collection, reporting, and project tracking, and is intended for collaborative software development projects. It works in a three-tier architecture: the client tier, the application tier and the data tier as displayed in figure 12.6. The *client tier* is used for creating and managing projects and accessing the items that are stored and managed for a project. TFS does not include any user interface for this tier, rather it exposes web services, located in the *application layer*, which client applications, like Visual Studio Team System, can use to integrate TFS functionality. The *application layer* also includes a web portal and a document repository facilitated by Windows SharePoint Services. The web portal, called the *Team Project Portal*, acts as the central point of communication for projects managed by TFS. The document repository is used for both project items and the revisions tracked, as well as for aggregated data and generated reports. The *data layer*, finally, provides the persistent data storage services for the document repository. It is not exposed to the *client tier*, only the *application tier* is.

Most activity in Team Foundation Server revolves around a *work item*. Work items are

⁸Based on http://en.wikipedia.org/wiki/Team_Foundation_Server

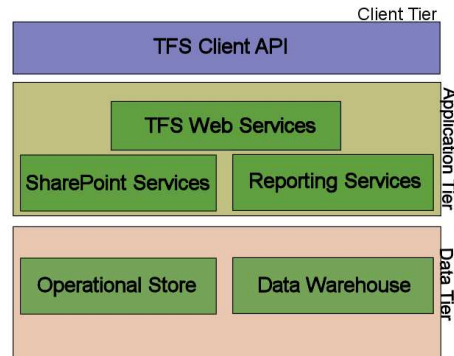


Figure 12.6: Team Foundation Server 3-tier architecture

a single unit of work which needs to be completed. It has fields to define area, iteration, assignee, reported by, a history, file attachments, and any number of other attributes. Work items themselves can be of several different types. The framework chosen for any given project in a Team Foundation Server defines what types of work items are available and what attributes each type of work item contains. When creating a project, a software development framework must be chosen, and cannot be changed afterwards. TFS includes a few common templates. Examples are *MSF for CMMI Process Improvement* which contains bugs, quality of service requirements, risks, scenarios and tasks, and *MSF for CMMI Process Improvement* which contains bugs, change requests, issues, requirements, reviews, risks and tasks. In this feasibility study we used the Scrum for Team Systems⁹ template, on which we will elaborate further below. TFS finally also provides a source control repository system called *Team Foundation Version Control* and extensive reporting capabilities.

*Scrum for Team Systems*¹⁰

As mentioned above, we use Scrum for Team Systems as the framework template for TFS in this feasibility study. Scrum for Team Systems defines the Product backlog items and the Sprint backlog items as Scrum related work items. A product backlog is a prioritized list of project requirements with estimated times to turn them into completed product functionality. Priority should be assigned based on the items of most value to the business or that offer the earliest Return on Investment. This list should evolve, changing as the business conditions or technology changes. Product backlog items can be functional requirements, non-functional requirements, and issues. The precision of the estimate depends on the priority and granularity of the Product Backlog item, with the highest priority items that can be selected in the first few Sprints being very granular and precise. The Sprint backlog is a list of tasks that defines a Team's work for a Sprint. The list emerges during Sprint planning. The tasks on the Sprint backlog are what the Team has defined as being required to

⁹<http://www.scrumforteamsystem.com>

¹⁰Based on <http://scrumforteamsystem.com/processguidance/v1/Artefacts/Artefacts.html>

turn committed Product Backlog items into system functionality. Each task identifies who is responsible for doing the work and the estimated amount of work remaining on the task on any given day during the Sprint. Scrum for Team Systems defines the following Scrum related reports:

1. **Sprint Burndown Chart.** Graph in which the vertical axis displays the hours of effort remaining for the Sprint and the horizontal axis displays the duration of the sprint in days. The burndown is supposed to be shown by the line of descent from the start of the Sprint with the starting hours, down to the end of the Sprint with no hours remaining. An example is shown in figure 12.7.

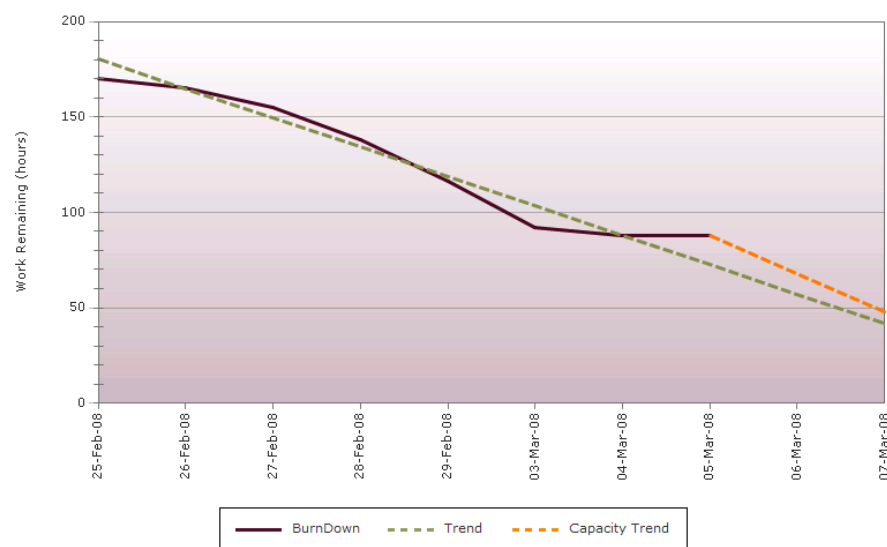


Figure 12.7: Sprint Burndown Chart

2. **Product Burndown Chart.** Comparable to the sprint burndown chart but applicable for the progress of completing product backlog items as opposed to sprint backlog items. An example is shown in figure 12.8.

TFS in connection with this feasibility study

In this feasibility study we use TFS to gather information to extend the user information further and also to incorporate project information. We extend the user information by extracting information regarding:

1. Which sprint backlog item each user is assigned to
2. What files each user currently has checked out
3. What files each user has last checked in



Figure 12.8: Product Burndown Chart

With respect to the project status we extract information regarding:

1. All releases per project
2. All sprints per release
3. All sprint backlog items per sprint
4. All product backlog items per project
5. All sprint backlog items per product backlog item
6. What files are currently checked out project wide
7. The last check-in of the entire project
8. The sprint burndown Chart
9. The product burndown Chart

12.4.2 Technical implementation

In this section we discuss the technical implementation of feasibility study. Our system consists out of four components.

1. *The OCS Data Collector*

This component collects presence information and information regarding the conversations from OCS and stores it in the database.

2. *The OC Actuator*

This component makes it possible for people not part of a certain conversation to request to join the conversation with the current leader of that conversation (initially the person starting the conversation).

3. *The TFS Data Collector*

This component collects information regarding projects, sprints, products and the code repository from TFS and stores it in the database.

4. *The User interface component*

This component displays the information we gathered in the OCS Data Collector and TFS Data Collector components in a combined way and offers the functionality we added in the OC Actuator component.

We will discuss these components in the rest of this section. Meanwhile we will introduce the database design on an as needed basis in the discussion of each component. We will conclude this section with an overview of the entire database design.

The OCS Data Collector

This component collects data from the Microsoft Office Communications Server. IHomer arranged for OCS accounts for us on the same server as they use themselves for us to test our implementation. Because they have the OCS server hosted by an external party it was infeasible for us to implement this server side. Because of this, we created a client side component which should run alongside all instances of Office Communicator (each and every OC user should run this component) and writes all relevant information to a MySQL database. This component uses the Communicator 2007 Automation API¹¹ to acquire the necessary information from Office Communicator.

This client side component consists of three subcomponents, a communicator component which gathers all data from OCS, a database component which retrieves and writes all information to the MySQL database and a main component which initializes and starts the database and communicator component. We wrote all these components in C# .NET because using this language the Communicator automation API can be called directly and we have a lot of experience with programming in Java, which is quite similar. Figure 12.9 shows how these components are interrelated.

¹¹<http://msdn.microsoft.com/en-us/library/bb758719.aspx>

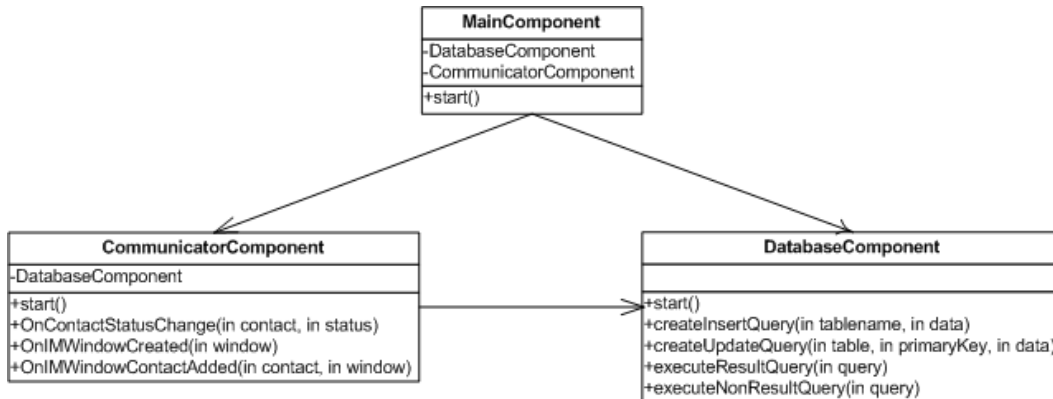


Figure 12.9: OCS class diagram

Communicator Component

As stated, the main objective of the communicator component is to collect data from OCS. This component gathers the required information two ways. On the one hand, it uses the Office Communicator Automation API to *actively* gather required information. On the other hand, it subscribes to receive notifications from this same API when certain events occur in communicator. When the communicator component is started it creates a list to be able to log the instant messaging conversations. This list contains references to all the instant messaging windows the user has started and a unique conversation id for each of them. It is the responsibility of this instance of the OCS data collector to log the history of these conversations in the database. Subsequently the component subscribes itself to receive notifications of the following events:

- OnContactStatusChange
- OnIMWindowCreated
- OnIMWindowContactAdded

Once the component is subscribed to these event handlers it checks if the user is logged in to communicator; if this is not the case it launches the sign in screen of communicator asking the user to log in. When the user is logged in, it updates the status of this user and the status of all other contacts to the database using the database component.

Now we have an instance of communicator running, which is subscribed to receive certain notifications, we are able to perform actions when one of the above mentioned events occur. Firstly we discuss the *OnContactStatusChange* event, which occurs when one of the contacts changes his or her status. When we receive a notification from the automation API that such an event has occurred we update the status of the associated user to the database using the database component. Another event we receive is the *OnIMWindowCreated* event,

which occurs when a conversation window is created. Upon receipt of this event the associated window reference should be added to the list of OCS window references discussed earlier if the user logged in on the computer where this instance of the data collector is running initiated the conversation. It is, however, not possible to directly determine from the event information whether or not this user has started the conversation himself. Therefore the window reference is temporarily stored in a variable and we wait until we receive a *OnIMWindowContactAdded* event. If the contact added in the first *OnIMWindowContactAdded* event received after the *OnIMWindowCreated* event is indeed the user of this OCS data collector instance himself, then he was the one that initiated the conversation and the window reference can be added to the list together with a unique conversation id, otherwise he was not and the window reference can be discarded.

Next to listening to notifications of the automation API the communicator component continuously executes a loop in which it *actively* uses this API to perform the following actions:

- Acquire the information regarding all users
- Acquire the information regarding the conversations

At the beginning of each loop we update the status of all users to the database, this includes basic contact information such as the sign in name, friendly name and the current status of a contact, this information can directly be obtained from a messenger contact using the Communicator API. These contacts also contain more detailed information about the status which can be gathered by using the presence properties of a contact. This information includes a note field, an indicator whether or not the contact is currently out of office and the time the contact is logged out. When all contact information is updated the information regarding the conversations from the reference list is updated in the database as well. This information includes the initiator, the start time, the history and the participants of the conversation. If a conversation in the list does not exist anymore it is removed from the associated list. When all this information is written to the database, we wait for a certain amount of time and restart the loop from the beginning.

Database Component

All the data gathered by the communicator component is written to the database using a database component. The structure of the tables in which the data is stored is shown in figure 12.10 and in figure 12.11.

12. PRACTICAL WORK

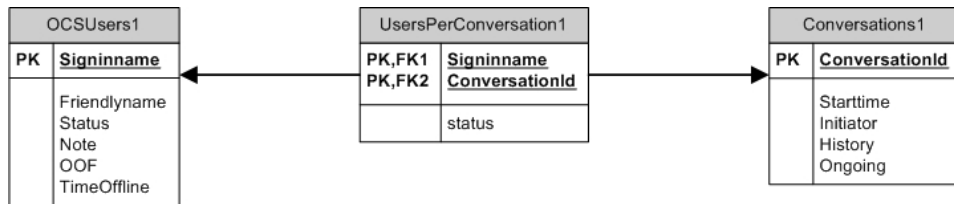


Figure 12.10: Relational Database Schema of the OCS Data collector

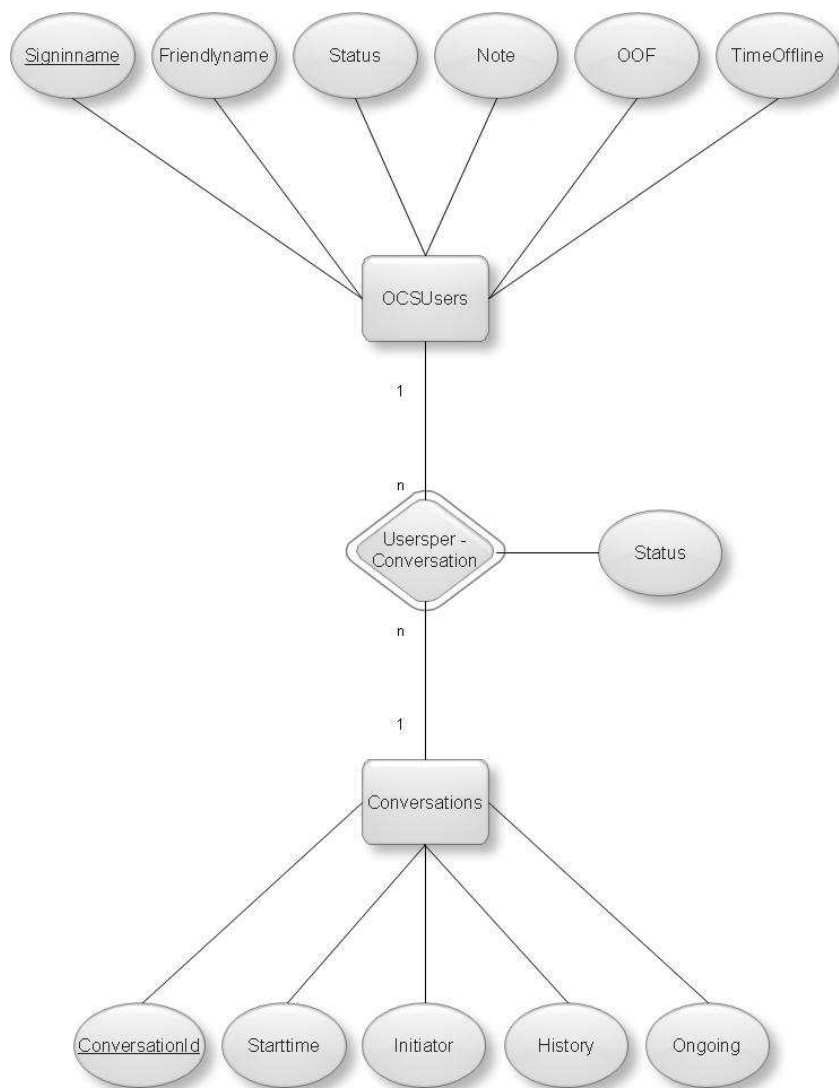


Figure 12.11: Entity Relationship Diagram of the OCS Data collector

The OC Actuator

This component makes it possible for people not part of a certain conversation to request to join the conversation with the current leader of that conversation (initially the person starting the conversation). Next to this, this component also makes it possible for the current leader of a conversation, the user that receives the join requests and is logging the conversation, to transfer ownership to another user when he closes the conversation. Because we implemented the OCS Data Collector client side anyway we extended this solution to incorporate this functionality. This component also uses the Communicator 2007 Automation API to acquire the necessary information from Office Communicator.

Communicator Component

We extend the communicator component defined in the OCS data collector section to incorporate the extra functionality. To make it possible for a user to join a conversation he is not part of, he should place a request to join the conversation into the database (using a user interface component). In the loop discussed in the OCS Data Collector component, functionality is added which checks for such requests for all conversations in the list. When such a request exists for a conversation in the list, the communicator component ask its user (automatically the leader) whether he wishes to allow this request. When the leader allows the request the user who placed the request is added to the conversation.

To make it possible to select a new leader, when the original initiator of a conversation leaves the conversation, we add a second list of references to conversation windows. In this list each instance of the OCS data collector stores the window-references of the conversations he did not initiate. We rename the original list to L1 and this second list to L2 to be able to differentiate between the lists. Next to this we further extend the functionality in the main loop of the communicator component. Firstly, when a conversation, this instance of OCS data collector is responsible for, is closed, its user is asked to select a new leader (using a pop-up screen). Subsequently, this owner switch request is placed in the database. Each OCS data collector instance checks this database table in its loop to determine if there are any outstanding requests for it to become a leader of a conversation. If this is the case, it moves the window-reference associated with that conversation from L2 to L1 and effectively takes over the logging and join-request handling responsibilities.

Database Component

All the data gathered by the communicator component is written to the database using the same database component as the OCS Data collector component. The structure of the tables is an extension of the tables of the OCS Data collector and are shown in figure 12.12 and in figure 12.13.

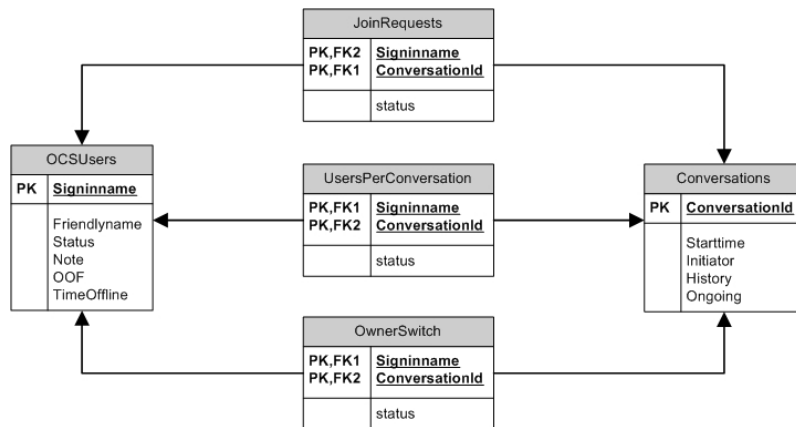


Figure 12.12: Relational Database Schema of the OC Actuator

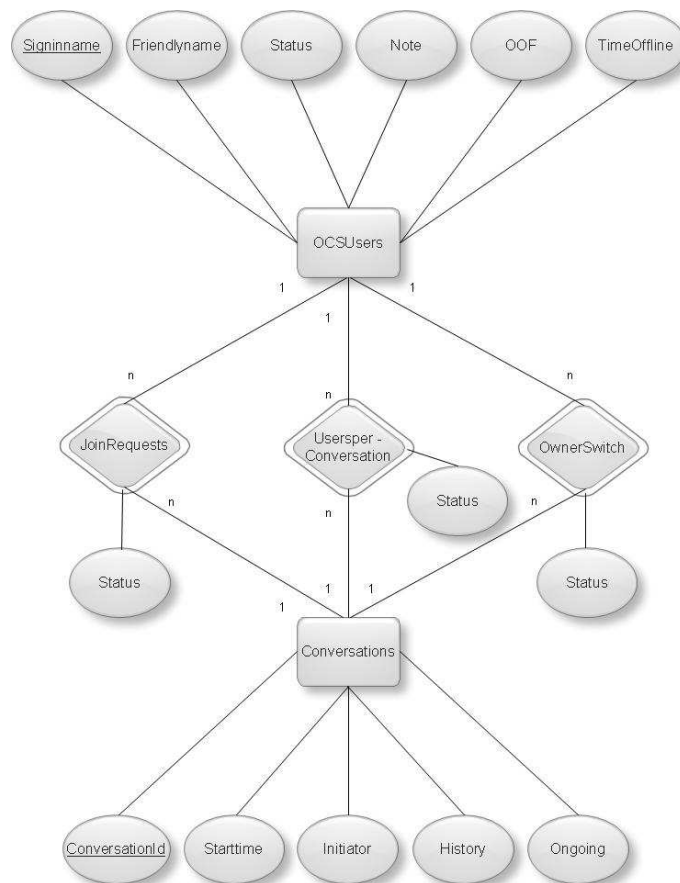


Figure 12.13: Entity Relationship Diagram of the OC Actuator

The TFS Data Collector

This component collects information regarding projects, sprints, products and the repository from TFS and stores it in the database. We host a trial version of Team Foundation Server ourselves, to be able to test our implementation. Because of this, we are able to develop this component server side.

This component also, like the OCS Data Collector, consists of three subcomponents: a TFS component which retrieves data from TFS, a database component which handles the communication with the database and a main component which initializes and starts the database and TFS component. Figure 12.14 shows how these components, all written in C#.NET, are interrelated. As discussed this component gathers different types of information

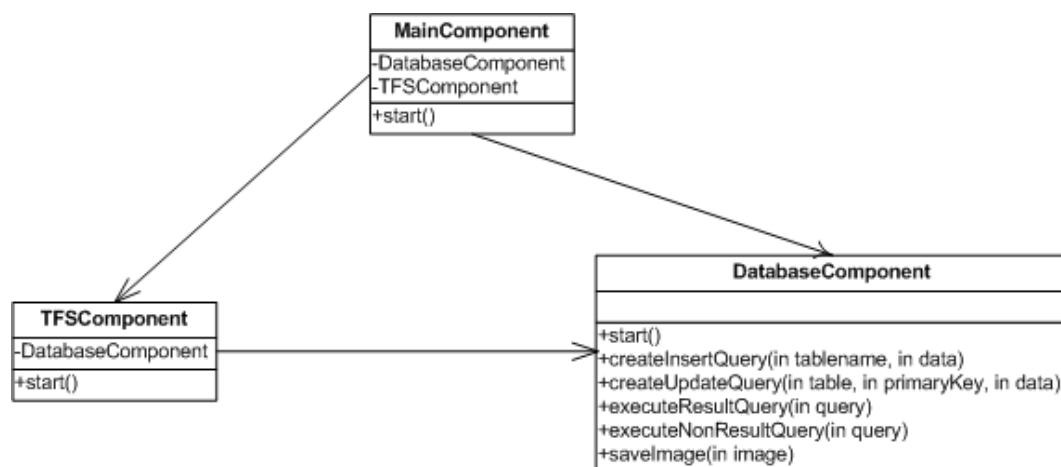


Figure 12.14: TFS Data Collector Component class diagram

from TFS. This is done by periodic, *active* retrieval of all required information from TFS via its associated APIs¹². We structure the discussion of the TFS Data Collector component by the types of information we gather:

- TFS Work Items
- TFS Version Control
- TFS Reports

We will discuss for each of these groups what data is collected and how the database is structured.

¹²[http://msdn.microsoft.com/en-us/library/bb130307\(VS.80\).aspx](http://msdn.microsoft.com/en-us/library/bb130307(VS.80).aspx)

TFS Work Items

The TFS Data Collector component can gather information about all projects which currently exist on the TFS. Firstly, relevant information for each of these projects, such as the project name and the project id, is gathered and written to the database. Secondly, data about the project iterations and associated information is gathered, this data includes releases, sprints and the iteration id. Subsequently all required information about a specific sprint is retrieved from TFS. Examples are: the title, the sprint id, the current state of the sprint, the start date, the end date and a short description. Finally information about all the product backlog items and sprint backlog items, associated with the current project, is gathered and stored in the database. We choose to gather the following information from the backlog items: the title, the description, state, estimated effort, work remaining and who is assigned to it. All the information mentioned above is gathered from different TFS Work Items. We use WIQL, a Work Item Query Language¹³, to be able to select the appropriate set of work items. All the information about the work items gathered by the TFS component is written to the database using the database component. The structure of the tables in which the data is stored is shown in figure 12.15 and in figure 12.16.

¹³<http://msdn.microsoft.com/en-us/library/bb130198.aspx>

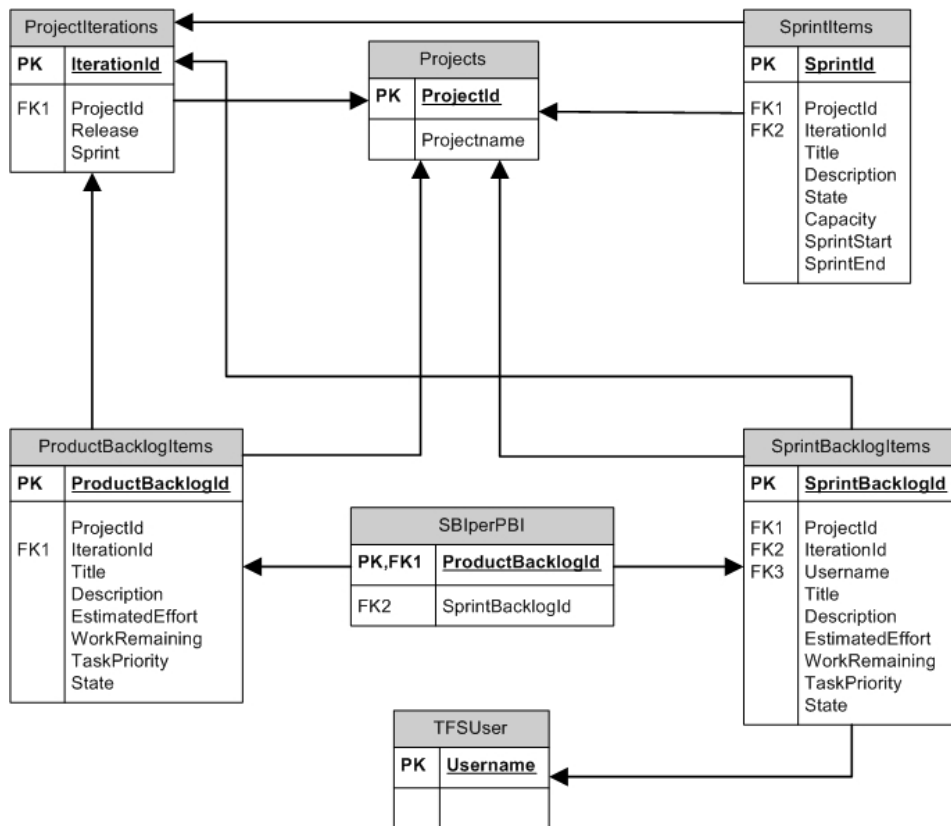


Figure 12.15: Relational Database Schema of the TFS Work Items

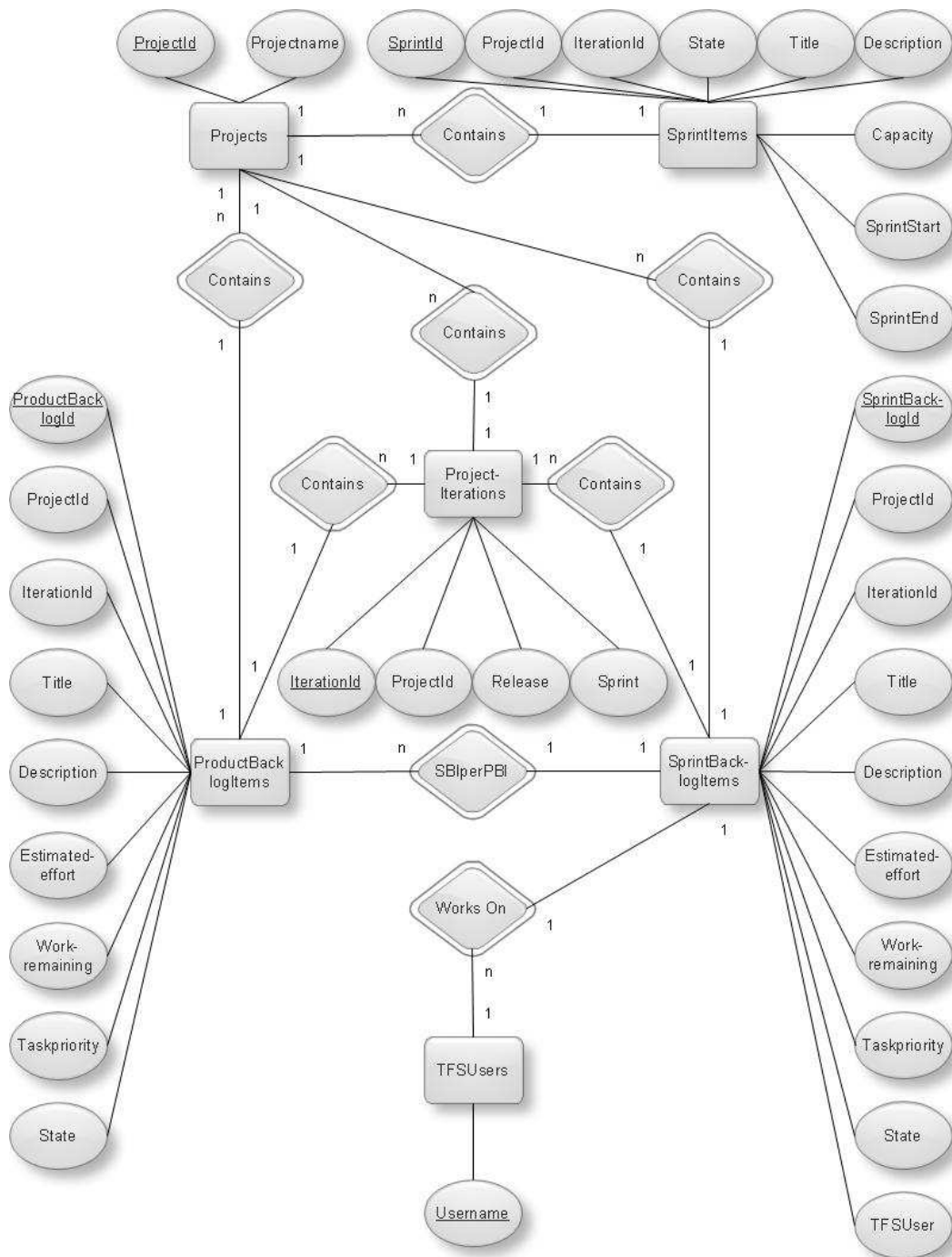


Figure 12.16: Entity Relationship Diagram of the TFS Work Items

TFS Version Control

This part of the TFS Data collector component gathers relevant information about TFS Version Control. It provides an overview of all the files a specific user currently has checked out. This overview is created by using the TFS Version Control API which makes it possible to retrieve a set of pending changes per project. This part also provides an overview of the last files a user has checked in into version control. This is done by selecting all change sets per project, after which the most recent change set per user is stored into the database. The structure of the tables in which the data, gathered by the TFS Data Collector component is stored is shown in figure 12.17 and in figure 12.18.

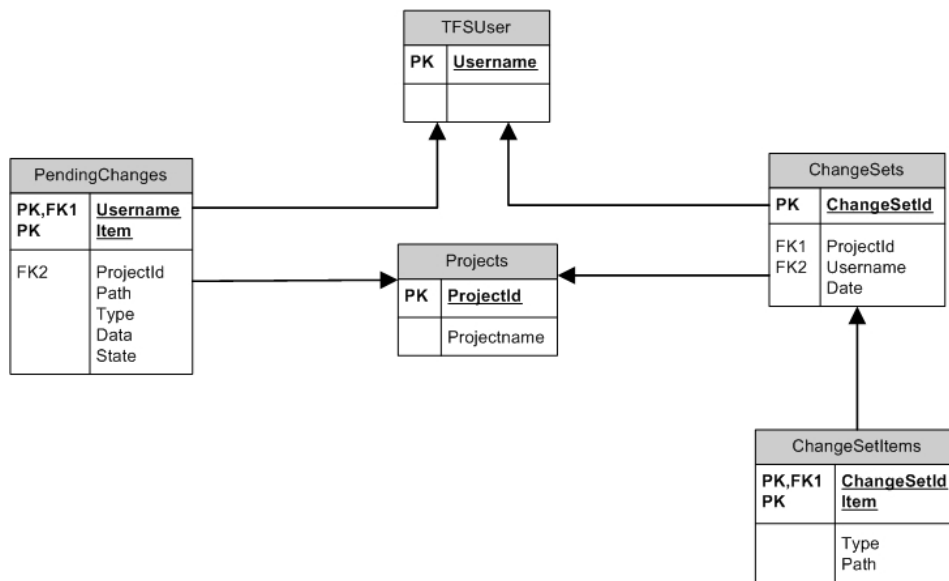


Figure 12.17: Relational Database Schema of the TFS Version Control

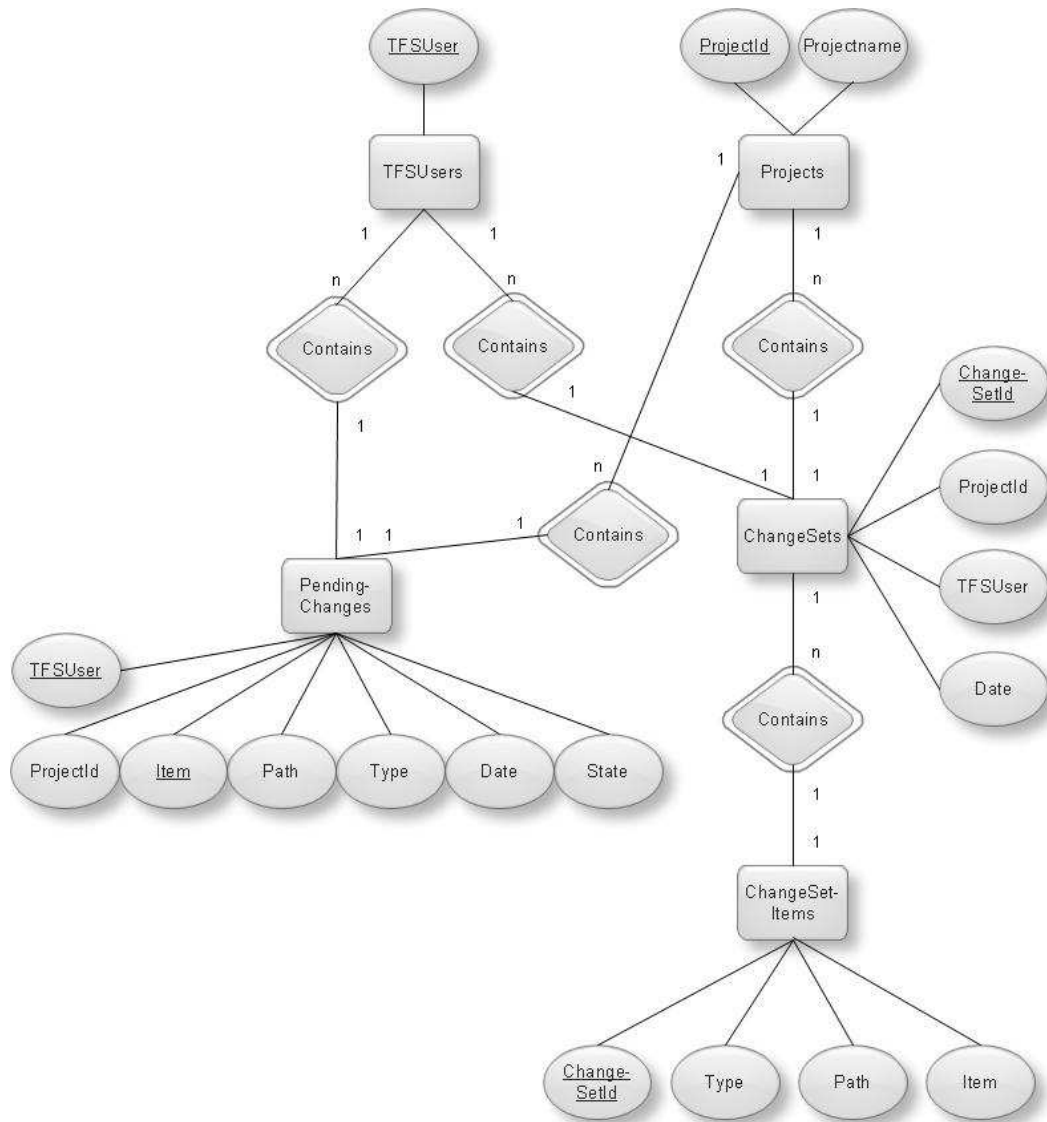


Figure 12.18: Entity Relationship Diagram of the TFS Version Control

TFS Reports

The last part of the TFS Data Collector component stores the generated reports of a TFS project into the database. An image file of these reports is gathered by performing an authenticated HTTP web-request to the project portal site. Subsequently this image is stored in the database in a BLOB; a binary large object. The structure of the tables in which the data is stored is shown in figure 12.19 and in figure 12.20.

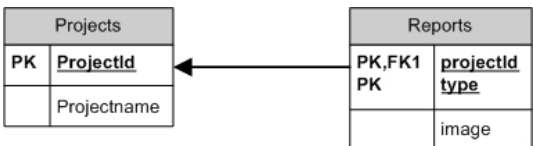


Figure 12.19: Relational Database Schema of the TFS Reports

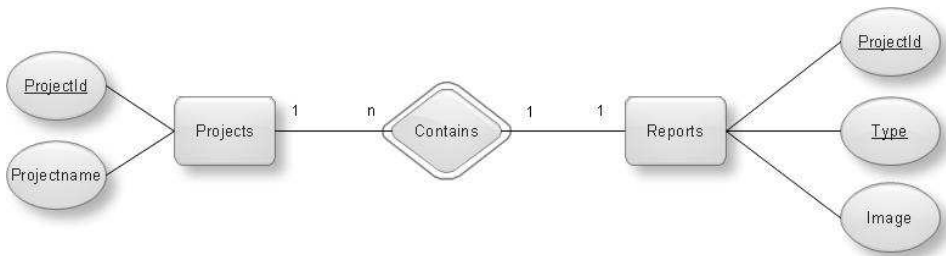


Figure 12.20: Entity Relationship Diagram of the TFS Reports

TFS Overview

In figure 12.21 and figure 12.22 an overview of the total database structure of the TFS Data Collector component is presented.

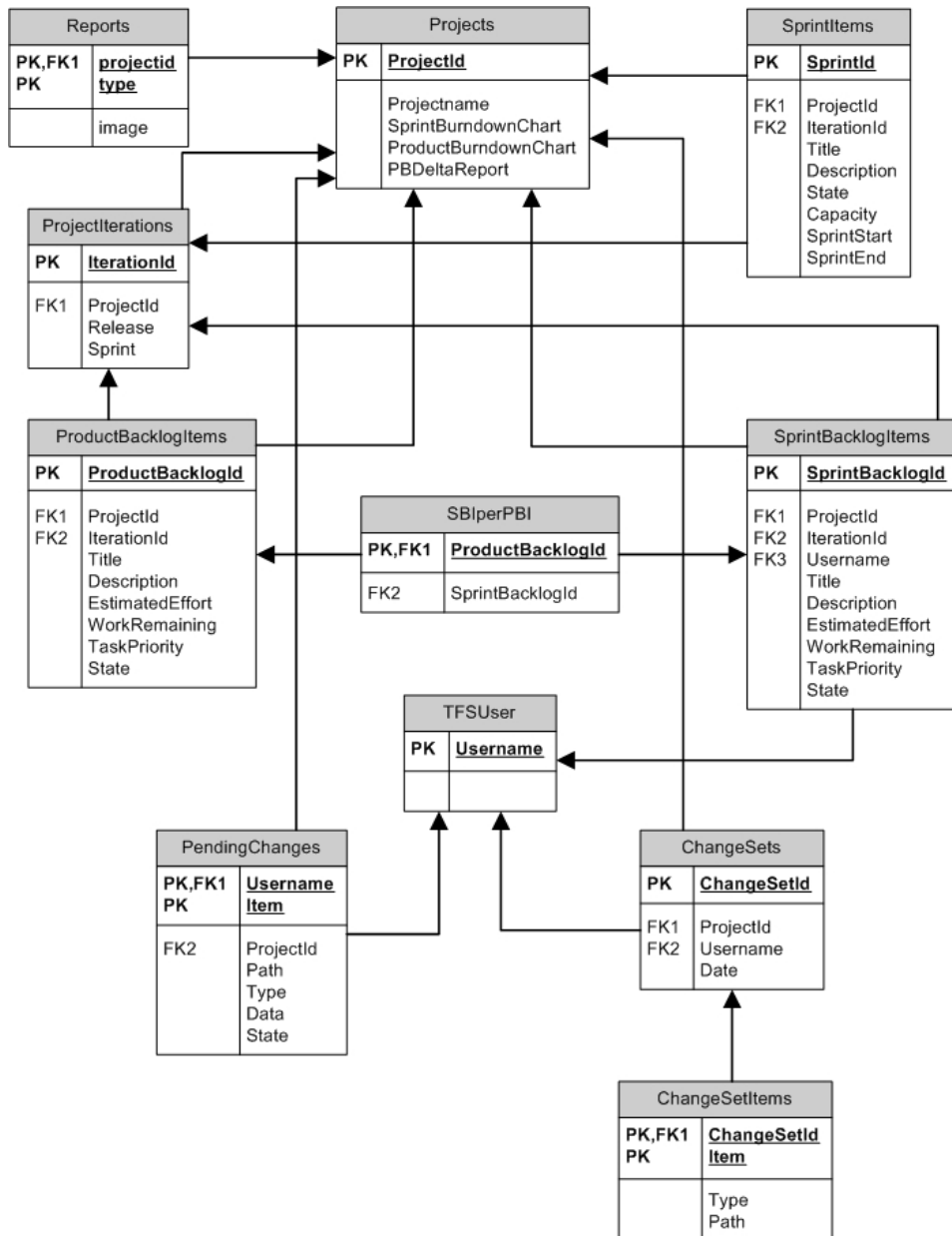


Figure 12.21: Relational Database Schema of the TFS Data Collector Component

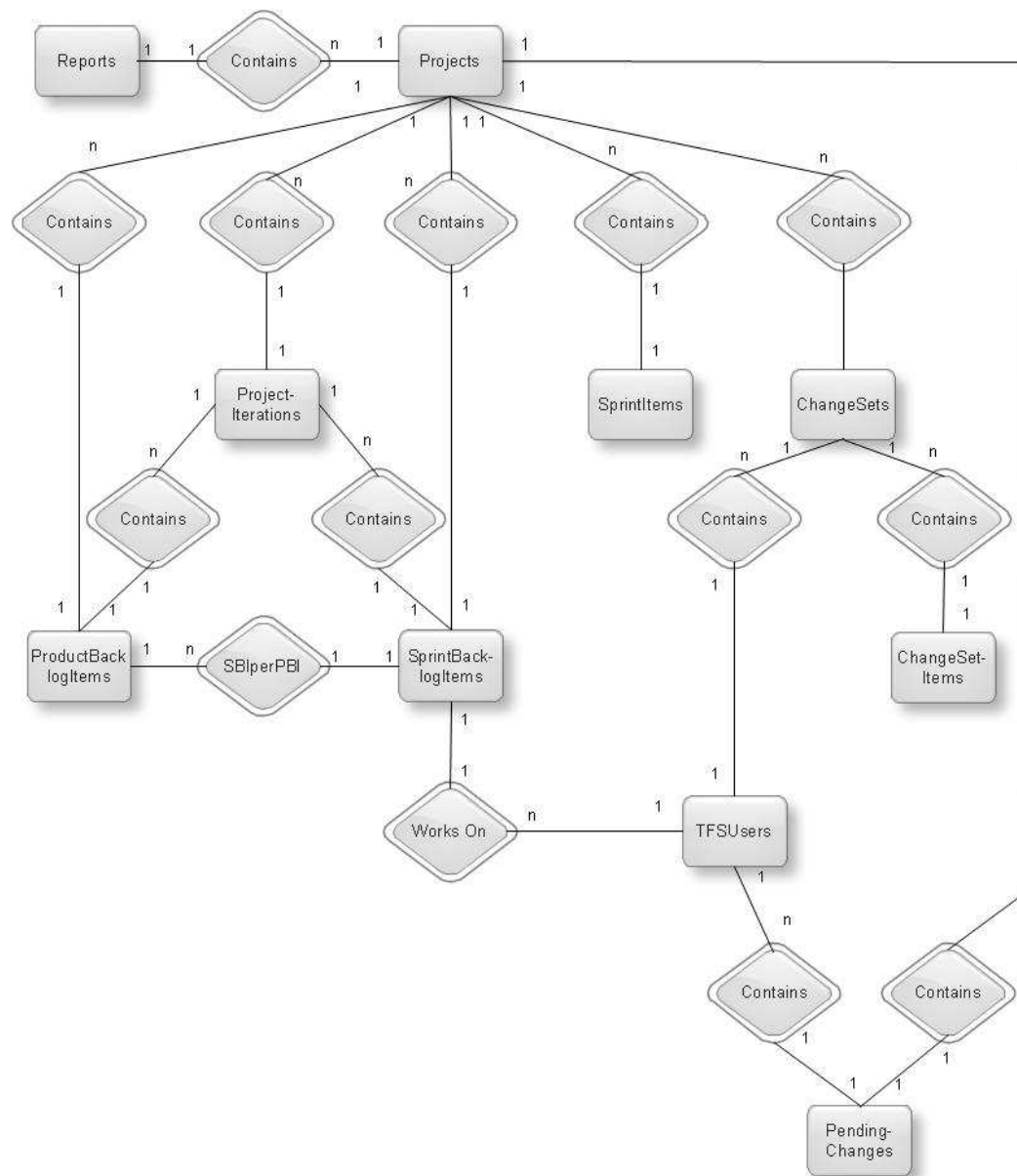


Figure 12.22: Entity Relationship Diagram of the TFS Data Collector Component

The User Interface Component

This component displays the information we gathered in the OCS Data Collector and TFS Data Collector components in a combined way and offers the functionality we added in the OC Actuator component. When the OCS Data Collector, the OC Actuator and the TFS Data Collector components are running and filling the database with data, several User Interfaces could be used at the same time. In this feasibility study we elected to create one such user interface in PHP generated HTML. We chose this technology because this allows us to show the functionality of our system in a separate application (Microsoft Internet Explorer 6 and Mozilla Firefox 3) but also because HTML pages can easily be integrated into Office Communicator. In our view, user interfaces that are integrated with the other development supporting technologies (like for example Visual Studio Team System) would prove useful as well, but are beyond the scope of a feasibility study.

For the technical implementation of the user interface we used AJAX (Asynchronous Javascript and XML) to load the required information and Thickbox to show specific information about an item. The overall page overview is shown in figure 12.23. When a menu item in the menu div is clicked the content of the specific page is loaded into the main div. The main div is then automatically reloaded every ten seconds with fresh data from the database. When an item about which our system contains information is clicked in the main div a Thickbox is displayed containing this information. A Thickbox is a webpage UI dialog widget written in JavaScript which function is to show inline content. It is opened on top of the existing page, mostly obscuring the underlying page. An example of this is shown in figure 12.24. Like the main div, the content of the Thickboxes is reloaded every ten seconds to provide an up-to-date overview of the data. When data items about which information is available in our system are clicked inside a Thickbox, the content of this is loaded in that same Thickbox. A complete discussion and overview of the user interface will be described in the User interface description section.



Figure 12.23: The design of the User Interface

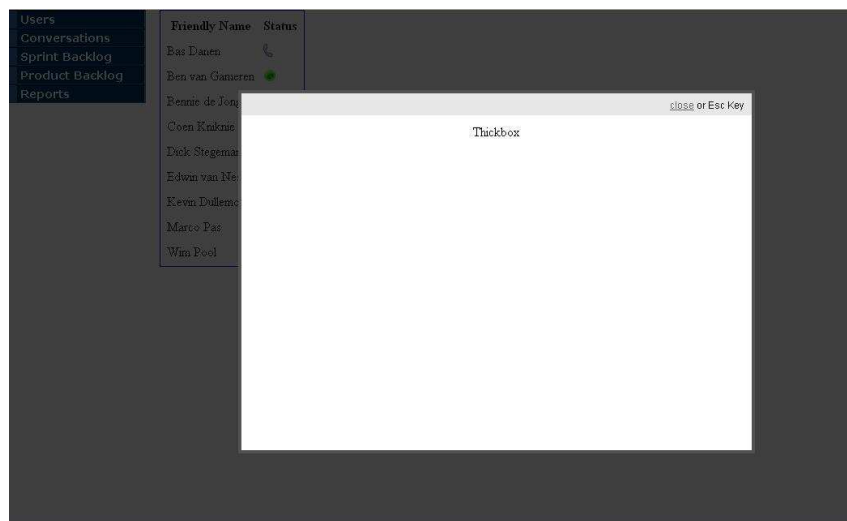


Figure 12.24: Example of a Thickbox

Database Overview

The total overview of the structure of the database is shown in figure 12.25 and in figure 12.26.

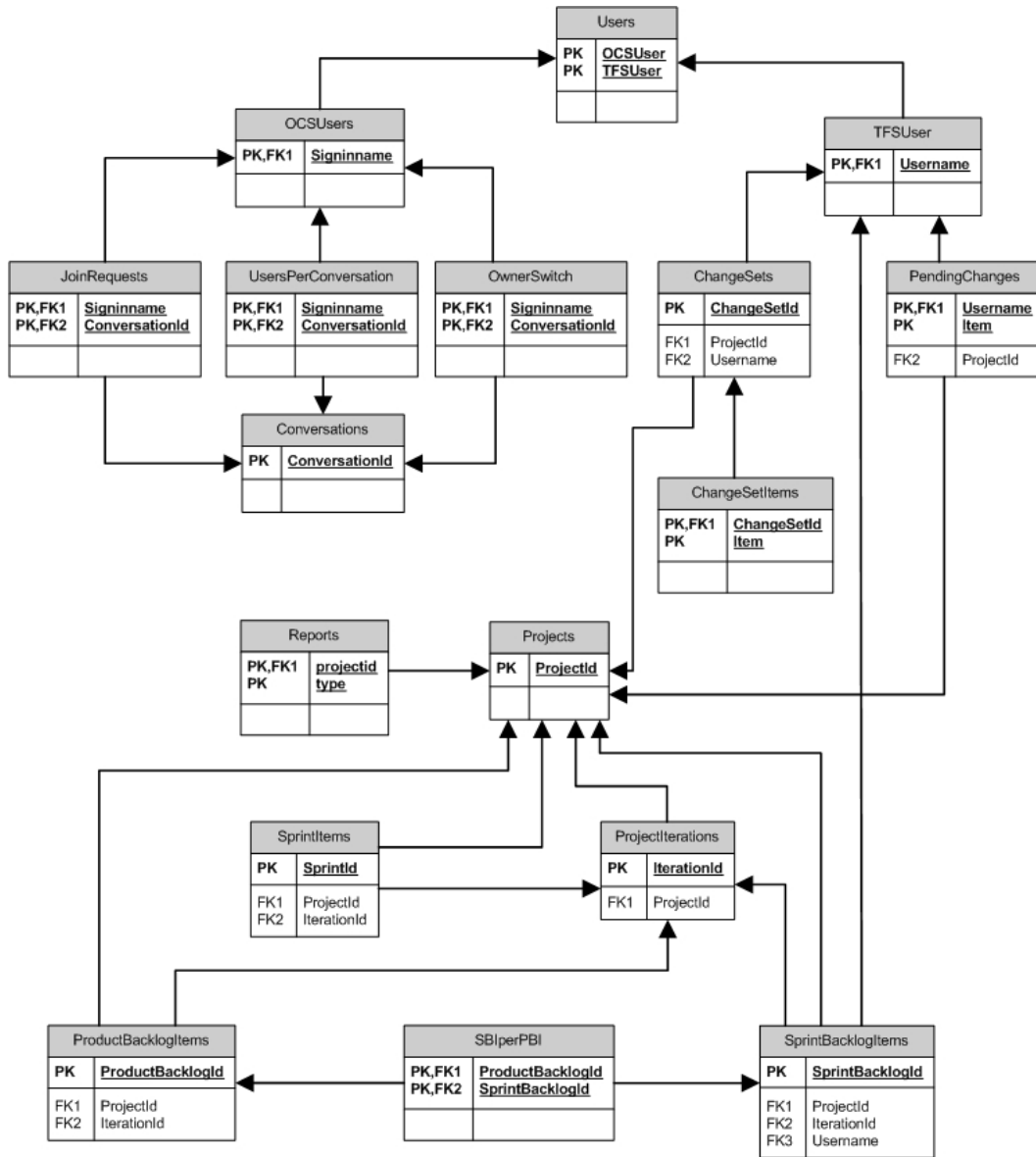


Figure 12.25: Relational Database Schema of the Total System

12.4.3 User interface description

In this section we will describe what information we display and how this is connected via menus and also what functionality is available in the user interface. To do this we will first explain the different pages the five menu items in the menu div evoke. For each of these pages we will provide a screenshot of an example, discuss all the information visible on the page and mention all the clickable data items. When such a data item is clicked a Thick-box is shown, as discussed in the previous section. Following the discussion of the menu items we will consecutively discuss each of these *Data Item Pages* with data by providing a screenshot, discussing all data visible in the box and what data items are shown on the page. We will conclude by discussing how we incorporated the functionality we added to OCS in our User Interface.

Menu

The menu bar is shown in figure 12.27. It consists of five entries which we will discuss consecutively.



Figure 12.27: The menu-bar of the User Interface

Users

Having clicked the *users* item in the menu bar a page like in figure 12.28 will be shown. On this page all the users will be displayed with their real name and their status in OCS. Clicking the name of a user will bring up the *user data item page*. The OCS status is dis-

Users	
Conversations	
Sprint Backlog	
Product Backlog	
Reports	





Friendly Name	Status
Bas Danen	●
Ben van Gasteren	●
Bennie de Jong	●
Coen Kniknie	●
Dick Stegeman	●
Edwin van Nes	●
Kevin Dullemond	●
Marco Pas	●
Wim Pool	●

Figure 12.28: The users list in the User Interface

played using the same icons as are used in Office Communicator. Hovering over this icon brings up a tooltip with a textual description of the current status of the respective user.

Conversations

On this page, shown in figure 12.29, the conversations held between the members of the development team using Office Communicator are shown. The conversations are divided in two groups: the conversations that are still active and the conversations that have already ended.

Users	Active conversations				
Conversations	Initiator	Start Time	Participant 1	Participant 2	Join
Sprint Backlog	 Ben van Gameren	2009-05-28 17:25:55	Ben van Gameren	Bennie de Jong	
Product Backlog	 Ben van Gameren	2009-05-28 17:25:05	Ben van Gameren	Edwin van Nes	
Reports					



Inactive conversations				
Initiator	Start Time	Participant 1	Participant 2	
 Ben van Gameren	2009-05-28 17:24:31	Bas Danen	Ben van Gameren	
 Ben van Gameren	2009-05-28 17:23:47	Ben van Gameren	Dick Stegeman	

Figure 12.29: The conversations list in the User Interface

The following information is shown regarding all conversations:

1. *A clickable history icon.* Clicking this icon brings up the *history data item page*.
2. *The initiator of the conversation.* Name of the user initially starting the conversation. Clicking this name will bring up the *user data item page*.
3. *The exact time the conversation has started.*
4. *The participants of the conversation.* Names of the users that are part of the conversation or have been part of the conversation in the past. Users that are still participating in the conversation are displayed in green, those that are no longer part of the conversation are displayed in red. Clicking one of these names will bring up the corresponding *user data item page*.

For the conversations that are still active an extra option is available when compared to the conversations that have ended. Behind each conversation a join-icon is shown. When this icon is clicked the *join a conversation* pages is displayed. How this functionality can be used is explained later when we explain how the functionality we added to OCS is incorporated in the User Interface.

Sprint Backlog

On this page a tree is shown with all projects as the root. Clicking a project name expands the respective project sub-tree to show all associated releases. Clicking a specific release in turn expands the tree further to show all sprints that are associated with the clicked release. Finally when a sprint is clicked a third and final expansion of the tree displays all sprint backlog items associated with the respective sprint. A screenshot of the totally expanded tree is shown in figure 12.30.

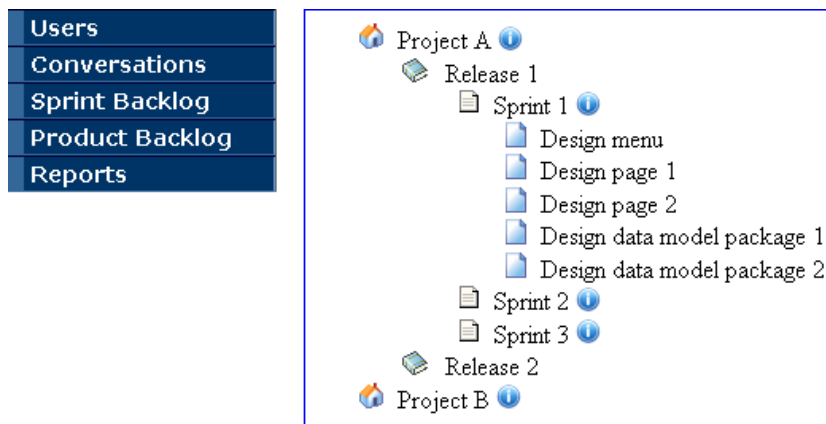


Figure 12.30: The Sprint Backlog

On this page links exist to the following *Data Item Pages*:

1. *The project data item page.* Is shown when the i-icon behind the respective project name is clicked.
2. *The sprint data item page.* Is shown when the i-icon behind the respective sprint name is clicked.
3. *The sprint backlog item data item page.* Is shown when the respective sprint backlog item is clicked.

Product Backlog

Like with the sprint backlog a tree is used to display the product backlog. As root nodes again the projects are used and when these are clicked the tree expands to show the associated product backlog items. The expanded tree is shown in figure 12.31. On this page two links exist to the following *Data Item Pages*:

1. *The project data item page.* Is shown when the i-icon behind the respective project name is clicked.
2. *The product backlog item data item page.* Is shown when the respective product backlog item is clicked.

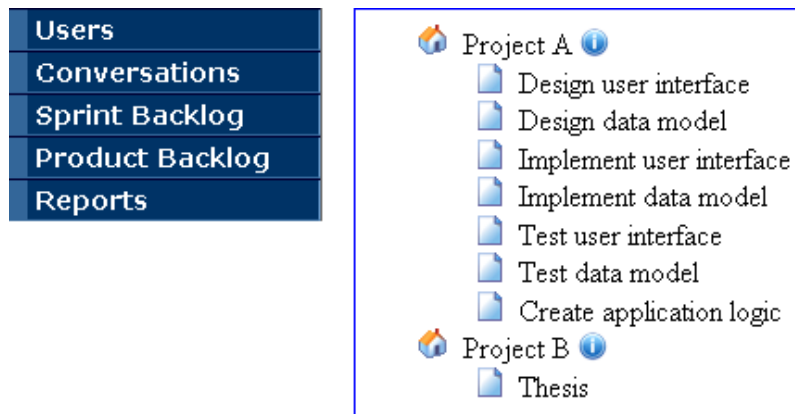


Figure 12.31: The Product Backlog

Reports

Like with the sprint backlog and the product backlog the reports are also accessed using a tree. As root nodes again the projects are used and when these are clicked the tree expands to show links to the two available reports per page. The expanded tree is shown in figure 12.32. On this page three links exist to the following *Data Item Pages*:

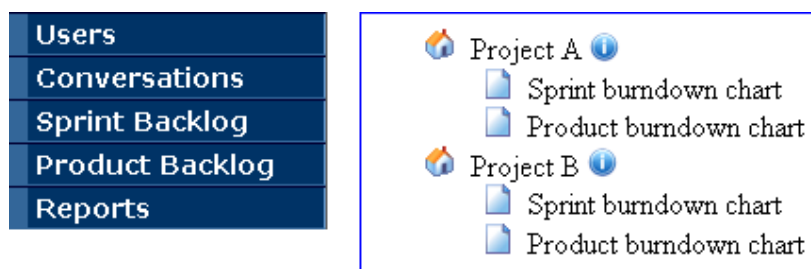


Figure 12.32: The Reports

1. *The project data item page.* Is shown when the i-icon behind the respective project name is clicked.
2. *The sprint burndown chart page.* Is shown when this option is clicked for a particular project.
3. *The product burndown chart page.* Is shown when this option is clicked for a particular project.

Data Item Pages

Below for all *Data Item Pages* we will provide a screenshot, discuss the information shown

on the page and discuss links it has to other *Data Item Pages*. Each of these *Data Item Pages* has a back and forward button at the top to ease navigating them.

User data

An example of this page is shown in figure 12.33.

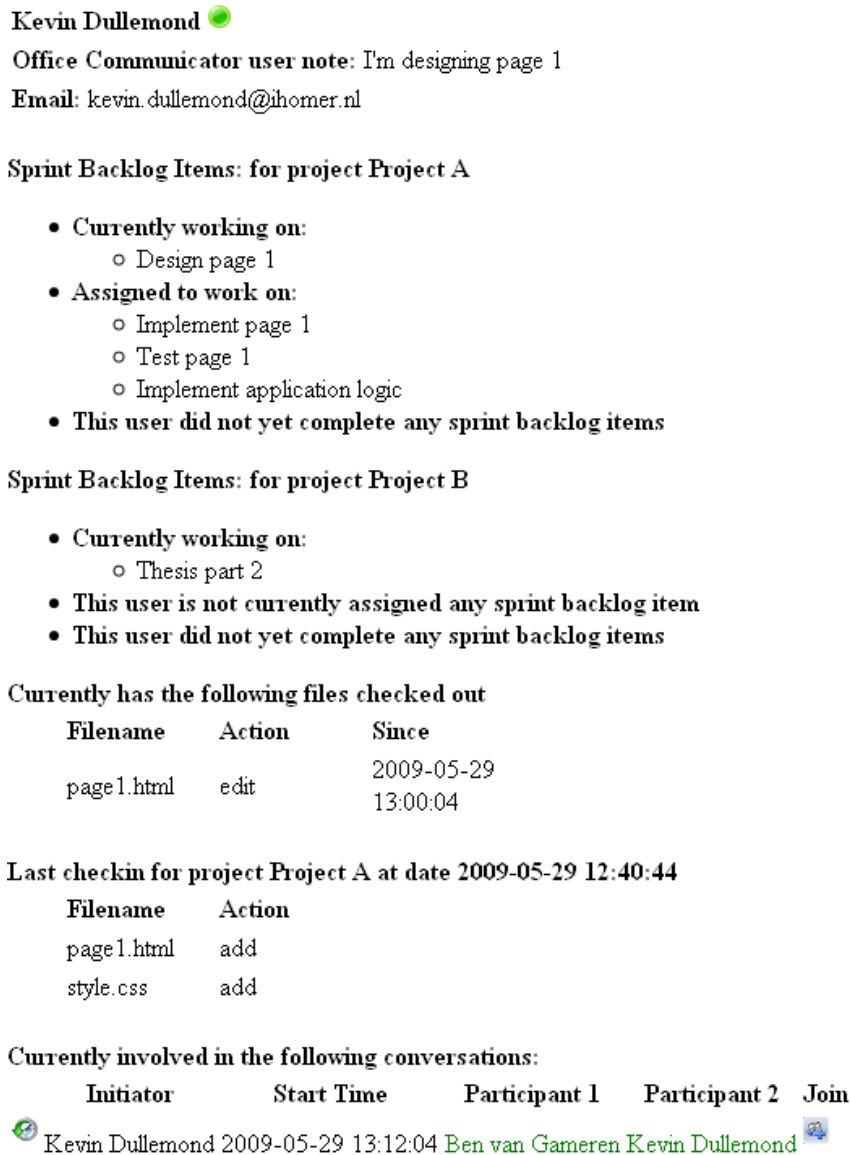


Figure 12.33: The User Data Item Page

This figure shows most of the available information with the exception of the notification

that the user is out of office and how long the user has been away (the user in this example is online). We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *Real name*. The real name of the user.
- *OCS status*. The OCS status is displayed using the same icons as are used in Office Communicator. Hovering over this icon brings up a tooltip with a textual description of this status.
- *Out of office*. If the user is out of office this information is displayed after the OCS status icon. This information is optional.
- *Office Communicator user note*. The text which the user has set in the note in office communicator. This information is optional.
- *Time away*. The total time the user has not been logged in to Office Communicator. This information is optional.
- *E-mail*. The email address used in Office Communicator.
- *Sprint backlog items*. This shows, per project, which sprint backlog items this user is associated with. These sprint backlog items are divided in three categories: The sprint backlog items that the user is currently working on, those that he is assigned to do but has not yet started to work on and those that he has already finished working on. In this overview clicking the project name links to the *Data Item Page* of the project and clicking the sprint backlog items links to the *Data Item Page* of that sprint backlog item.
- *Currently checked out files*. Shows what files this user currently has checked out, what action he has performed on them and since when each file is checked out.
- *Last checkin*. Shows what files this user has last checked in, when this check-in took place, what action was performed on each file and what project the check-in is associated with. In this overview clicking the project name links to the *Data Item Page* of the project.
- *Conversations currently involved in*. Displays a list of all active conversations this user is involved in. This overview contains precisely the same information and links as the active conversations in the conversations menu page discussed earlier.

History

An example of this page is shown in figure 12.34. We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *The initiator of the conversation*. Name of the user initially starting the conversation. Clicking this name links to the *user data item page*.
- *The exact time the conversation has started*.

Initiator: Kevin Dullemond
Start time: 2009-05-29 13:12:04
Participants:
Ben van Gameren
Kevin Dullemond

History

Kevin Dullemond [1:05 PM]:
Hello Ben
Ben van Gameren [1:05 PM]:
Hi Kevin, can I help you?
Kevin Dullemond [1:07 PM]:
yes, I'm currently working on Sprint Backlog Item Design page 1 and I'm uncertain on what to do next

Figure 12.34: The History Data Item Page

- *The participants of the conversation.* Names of the users that are part of the conversation or have been part of the conversation in the past. Users that are still participating in the conversation are displayed in green, those that are no longer part of the conversation are displayed in red. Clicking one of these names links to the corresponding *user data item page*.
- *The exact text the participants of the conversation have said in the conversation.*
Note: Like all data on all *Data Item Pages* also the conversation is updated automatically so it is possible to passively follow the conversation.

Project

An example of this page is shown in figure 12.35. We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *The name of the project.*
- *Currently checked out files.* Shows what files are currently checked out for this project, what action is performed on them, since when each file is checked out and what user has checked out each file. Clicking the name of one of these users links to the *user data item page* of that user.
- *Last checkin.* Shows what files are last checked in for this project, when this check-in took place, what user performed the check in and what action was performed on each file. Clicking the name of the user that performed the check in links to the *user data item page*.
- *Product backlog items.* This shows what product backlog items are associated with this project. In this overview clicking the product backlog items links to the *Data Item Page* of that product backlog item.

Project: Project A

Currently, the following files are checked out

Filename	User	Action	Since
page1.html	Kevin Dullemond	edit	2009-05-29 13:00:04

Last checkin by user Kevin Dullemond at date 2009-05-29 12:40:44

Filename	Action
page1.html	add
style.css	add

Product Backlog Items

- Design user interface
- Design data model
- Implement user interface
- Implement data model
- Test user interface

.....

Sprint Backlog items

- Design menu
- Design page 1
- Design page 2
- Design data model package 1
- Design data model package 2

.....

Figure 12.35: The Project Data Item Page

- *Sprint backlog items.* This shows what sprint backlog items are associated with this project. In this overview clicking the sprint backlog items links to the *Data Item Page* of that sprint backlog item.

Sprint

An example of this page is shown in figure 12.36. We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *The name of the sprint.*
- *A description of the sprint.*
- *The capacity in hours of the sprint.*
- *The exact start date of the sprint.*
- *The exact end date of the sprint.*
- *The current state of the sprint.*
- *Part of project.* The name of the project this sprint is part of. Clicking this name links to the associated project data item page.

Sprint: Sprint 1
Description: Design everything
Capacity: 50
Start-date: 5/28/2009 6:00:35 AM
End-date: 6/8/2009 6:01:01 AM
State: In Progress
Part of project: Project A

Product Backlog Items

- Design user interface
- Design data model

Sprint Backlog items

- Design menu
- Design page 1
- Design page 2
- Design data model package 1
- Design data model package 2

Figure 12.36: The Sprint Data Item Page

- *Product backlog items.* This shows what product backlog items are associated with this sprint. In this overview clicking the product backlog items links to the *Data Item Page* of that product backlog item.
- *Sprint backlog items.* This shows what sprint backlog items are associated with this sprint. In this overview clicking the sprint backlog items links to the *Data Item Page* of that sprint backlog item.

Sprint backlog item

An example of this page is shown in figure 12.37. We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *The title of the sprint backlog item.*
- *A description of the sprint backlog item.*
- *The estimated hours for the sprint backlog item.*
- *The number of hours remaining until completion.*
- *The priority of this sprint backlog item.*
- *The current state of the sprint backlog item.*
- *Assigned to.* Name of the user to which this sprint backlog item is assigned. Clicking this name links to the *user data item page*.
- *Part of project.* Name of the project this sprint backlog item is part of. Clicking this name links to the respective *project data item page*.

Title: Design menu
Description: Design the menu for the system
Estimated Effort: 10
Work remaining: 10
Task Priority: 1
State: In Progress
Assigned to: Ben van Gasteren
Part of project: Project A
Part of sprint: Sprint 1

Product Backlog Items:

- Design user interface

Figure 12.37: The Sprint Backlog Item Data Item Page

- *Part of sprint.* Name of the sprint this sprint backlog item is part of. Clicking this name links to the respective *sprint data item page*.
- *Product backlog items.* This shows what product backlog item this sprint backlog item is associated with. Clicking the product backlog item links to the *Data Item Page* of that product backlog item.

Product backlog item

An example of this page is shown in figure 12.38.

Title: Design user interface
Description: I want a designed UI
Estimated Effort: 25
Work remaining: 20
Task Priority: 1
State: In Progress
Part of project: Project A
Part of Sprint: Sprint 1
Sprint Backlog Items:

- Design menu
- Design page 1
- Design page 2

Figure 12.38: The Product Backlog Item Data Item Page

We will now give an overview of the information shown on the page and discuss links it has to other *Data Item Pages*:

- *The title of the product backlog item.*
- *A description of the product backlog item.*
- *The estimated hours for the product backlog item.*
- *The number of hours remaining until completion.*
- *The priority of this product backlog item.*
- *The current state of the product backlog item.*
- *Part of project.* Name of the project this product backlog item is part of. Clicking this name links to the respective *project data item page*.
- *Part of sprint.* Name of the sprint this product backlog item is part of. Clicking this name links to the respective *sprint data item page*.
- *Sprint backlog items.* This shows what sprint backlog items this product backlog item is associated with. Clicking the product backlog items links to the respective *Data Item Page* of that product backlog item.

Sprint burndown chart

An example of this page is shown in figure 12.39.

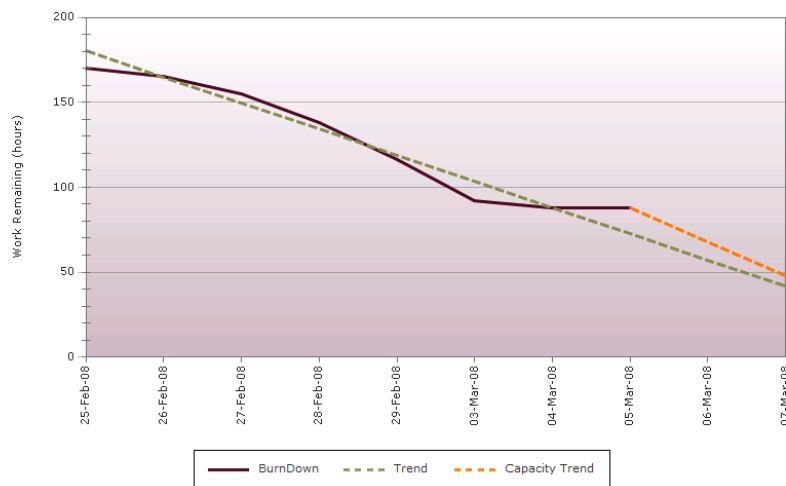


Figure 12.39: The Sprint Burndown Chart

This page only shows the sprint burn down chart.

Product burndown chart

An example of this page is shown in figure 12.40.



Figure 12.40: The Product Burndown Chart

This page only shows the product burn down chart.

Added functionality

Up till now our system has mainly combined and visualized existing information from multiple sources. However, additional functionality has been added as well. Our system provides the possibility of requesting to join an ongoing conversation in Office Communicator. To do this click on the join-icon behind the conversation you want to join, either on the conversations menu page or the Thickbox of a specific user. When you click this button a window such as the one shown in figure 12.41. In this screen you should type your OCS username and click join. Doing so leads to the screen shown which shows the current state of your join-request. While you are watching this screen the user that originally started the conversation gets a pop up stating you are requesting to join the conversation, asking whether or not he wants to allow this. A screenshot of this popup box is shown in figure 12.42. If the user chooses to allow your request the status of the request is updated to *approved* on the page and you will soon be added to the conversation. If the user chooses to decline your request the status of your request is updated to *declined* on the status page. If your request to join a certain conversation is declined once, it is not possible to request to join that conversation again as it will automatically be declined.

Do you want to join this conversation?

Username:

Figure 12.41: The join screen

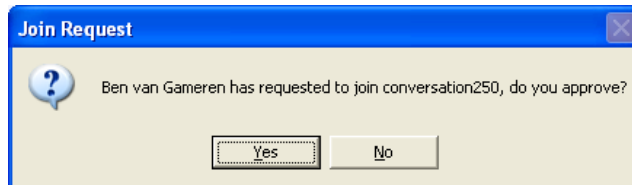


Figure 12.42: The incoming join request pop-up

A final part of our system is the possibility to elect a new leader of the conversation when the original leader leaves the conversation while others continue the conversation. To accomplish this, when the user originally starting the conversation closes the conversation window when at least two other users are still active in the conversation a screen such as the one shown in figure 12.43 is popped up asking this user to select one of the other users to take over the leadership. When he has selected one of the other users and clicked apply the selected user will be notified that he is the new leader of the conversation with a pop-up box shown in figure 12.44. From now on this user will receive the request of people asking to join the conversation. When this user closes the conversation when at least two users are still active in the conversation he, in turn, will have to elect one of these as the new leader.



Figure 12.43: The popup screen to select a new leader



Figure 12.44: The pop-up screen notification that you are the new leader of a conversation

12.4.4 Validation of the feasibility study

With the system completed we returned to the three companies which took part in the definition of the feasibility study, to confirm the technical feasibility of the three concepts we implemented. For each of these companies we presented our system to a representative and requested feedback both regarding the technical feasibility and also regarding their views of further expansion of our system. All three companies agreed this system confirms the technical feasibility of the three concepts we attempted to implement. Therefore we conclude this feasibility study has shown the technical feasibility of the three concepts. The companies particularly liked how the information from both systems was linked together and the increased visibility of the conversations of remote colleagues they would otherwise have missed. Finally, we discussed a number of ideas for further expansion of the system:

1. Increasing the scalability of the user interface by clustering data and providing the possibility to filter the data based on certain parameters.
2. Allowing the users to have private conversations by adding the possibility for users to switch between public and private conversations.
3. Annotating the conversations in a fashion so the content of the conversation can be seen more easily in a single glance. This can for instance be achieved by showing the last line of a conversation or a tag-cloud of the content of the conversation, for each conversation in the conversations list.
4. Creating user interfaces for each of the programs used often in the development process, for example Visual Studio.
5. Extending the type of systems we use as a source of information. Examples of such additional systems are issue management systems, build servers, Enterprise Resource Planning systems, Enterprise Relationship Management systems, Customer Relationship Management systems and workflow systems.
6. Extending the information that can be deduced by our system by use of artificial intelligence. By doing this, the system could for instance proactively notify the users of information that could be useful to them.

12.5 Validity of the practical work

In this section we will reflect on the validity of the entire practical research. The first consideration with respect to the validity concerns the construct validity. Construct validity of a research means that the research is constructed in a fashion that it develops a sufficient operational set of measures and ensures that subjective judgments are used to collect the data [143]. To meet these requirements we explicitly defined a research design. In this research design we described how we were going to collect and analyze the data as precisely as possible and had this research design reviewed by the supervisor of this research. Another tactic we use to be able to claim construct validity is using multiple sources of evidence because a finding or conclusion from a study is much more convincing and accurate if it is based on several different sources of information. In fact, we used triangulation by first performing document based research (our literature research) and subsequently gathering data by performing several interviews. The second consideration with respect to the validity of the research concerns internal validity. Internal validity of research has to do with showing that inferences made in the research are valid by properly demonstrating a causal relation between them. In this research however, we do not try to infer matters, but to elicit a list of requirements for an ICDE to support distributed agile development and a number of concepts which implement these. Therefore internal validity is not applicable to our research. We also cannot claim external validity, the third consideration which concerns the generalizability of the findings of a study, outside of the domain our study was carried out in. This is because generalization of findings outside the domain the study was carried in is not automatic. In order to generalize the results of our research, from the domain of five companies part of our knowledge group to the domain of all software engineering companies which develop globally distributed using agile methodologies, this research should be replicated in a few different domains and produce similar results. We have not performed such external replications as of yet and thus cannot claim external validity. We have however done something similar to replication research but on a smaller scale within the research itself, by asking the companies to validate each others results. This does not directly influence external validity, yet it does improve the results. The fourth and final consideration with respect to the validity of the research concerns whether or not a research is reliable. Reliability of research means that if the research is repeated it will produce exactly the same results. We cannot claim our research is reliable. This is because a large portion of the data is acquired by semi-structured interviews which, by definition, cannot be repeated in exactly the same fashion.

12.6 Conclusions and recommendations for further research

In this chapter we described the research we performed based on a recommendation made in part V of this thesis. In that section we discuss supporting agile GSD with technology by integrating the supporting technologies into an Integrated Collaborative Development Environment (ICDE). In this chapter we have attempted to make a first step into the construction of such an ICDE by defining a categorized list of requirements of an ICDE, deducing a number of concepts to materialize these requirements and finally showing the technical fea-

sibility of three of these concepts by implementing these in a feasibility study.

In this section we will discuss the recommendations of further research we have after conducting the research described in this chapter. We will not reiterate the recommendations voiced in part V of the research regarding the more theoretical part of our research. These recommendations still stand, but here we describe what further research we recommend regarding the creation of an ICDE. In other words: we will discuss how we think the specific line of research started in this chapter should be continued.

To start, we argue the line of research discussed in this chapter is viable and further research is warranted. We draw this conclusion based on the experiences we had during this past year with the companies in our research group, the literature research we performed and the practical research we did. In our opinion the distances faced when developing globally distributed can be more easily dealt with, when a system is in place which essentially does two things. Firstly, it integrates all work related information and effectively provides this to the users to support the exchange of knowledge and avoid misunderstanding. Secondly, it helps make remote communication more natural and intuitive and thus supports the communication process.

Having said this, there is still a lot of work to be done before a system as described above can be realized. For one, the list of requirements should be validated and possibly extended by performing research similar to ours with another set of companies. When the list of requirements reaches a state where subsequent research no longer leads to large extensions or alterations, external validity can be claimed for the domain the companies are elected from. Subsequently, the concepts we defined to realize certain requirements should be researched further. Firstly, subsequent validation is required to be certain the concepts actually support distributed development. Examples of this are cases studies in which these concepts are tried out and subsequently evaluated by investigating work items and performing interviews with the people involved. Secondly the concepts themselves should be investigated further as well. In our system for example we elected to gather certain types of information about users and the project from two particular systems. Research in this area would concern deducing what information is useful and from which systems this information can be gathered. Thirdly, we did not show the technical feasibility of a "mandatory" open line. A feasibility research should be done to show this as well. Finally, altogether new concepts could be defined to realize particular requirements of the requirement list.

Bibliography

- [1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile software development methods. Review and analysis*. VTT Publications, 2002.
- [2] A. Abran and J.W. Moore. *Swebok: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, 2004.
- [3] P.J. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, and E.Ó. Conchúir. Benefits of global software development: The known and unknown. In Q. Wang, D. Pfahl, and D.M. Raffo, editors, *ICSP*, volume 5007 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2008.
- [4] P.J. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, B. Lings, B. Lundell, and E.Ó. Conchúir. A framework for considering opportunities and threats in distributed software development. In *In Proceedings of the International Workshop on Distributed Software Development (DiSD 2005)*, pages 47–61, August 2005.
- [5] T. Allen. *Managing the flow of technology*. MIT press, 1977.
- [6] M. Aoyama. Web-based agile software development. *Software, IEEE*, 15(6):56–65, Nov/Dec 1998.
- [7] A. Arora and S. Athreye. The software industry and india’s economic development. *Information Economics and Policy*, 14(2):253–273, June 2002.
- [8] M.A. Awad. *A Comparison between Agile and Traditional Software Development Methodologies*. Honours program thesis, University of Western Australia, 2005.
- [9] S. Baker, G. McWilliams, and M. Kripalani. Forget the huddled masses: Send nerds. *Business Week*, pages 110–116, July 1997.
- [10] R. Barner. The new millennium workplace: Seven changes that will challenge managers and workers. *The Futurist*, 30:14–18, 1996.
- [11] R.D. Battin, R. Crocker, J. Kreidler, and K. Subramanian. Leveraging resources in global software development. *Software, IEEE*, 18(2):70–77, Mar/Apr 2001.

- [12] B Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.
- [13] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.
- [14] K. Beck. *Test Driven Development: By Example*. Addison-Wesley, 2003.
- [15] K. Beck. *Extreme Programming Explained: Embrace Change, Second edition*. Addison-Wesley, 2004.
- [16] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. The agile manifesto. *The agile alliance*, 2001.
- [17] R.E. Berry and B.A.E. Meekings. A style analysis of c programs. *Commun. ACM*, 28(1):80–88, 1985.
- [18] M. Berteig. Agile is not communism. 2007.
- [19] L.L. Bierema, J.W. Bing, and T.J. Carter. The global pendulum. *Training and Development*, 56(70):72–78, may 2002.
- [20] G.L. Bodic. *Mobile Messaging Technologies and Services: SMS, EMS and MMS*. John Wiley & Sons, Inc., 2005.
- [21] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.
- [22] B. W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, pages 61–72, May 1988.
- [23] B.W. Boehm. Get ready for agile methods, with care. *IEEE Computer*, 35(1):64–69, 2002.
- [24] D. Boland and B. Fitzgerald. Transitioning from a co-located to a globally-distributed software development team: a case study at analog devices inc. *IEE Seminar Digests*, 2004(912):4–7, 2004.
- [25] C. Boyapati, S. Khurshid, and D. Marinov. Korat: automated testing based on java predicates. In *International Symposium on Software Testing and Analysis (ISSTA '02)*, pages 123–133, July 2002.
- [26] E. Bradner, G. Mark, and T.D. Hertel. Team size and technology fit: participation, awareness, and rapport in distributed teams. *Professional Communication, IEEE Transactions on*, 48(1):68–77, March 2005.

-
- [27] K. Braithwaite and T. Joyce. Xp expanded: Distributed extreme programming. In *Extreme Programming and Agile Processes in Software Engineering*, pages 180–188. Springer Berlin / Heidelberg, 2005.
- [28] M.A. Bunge. *Treatise on Basic Philosophy: Volume 4: Ontology II: A World of Systems*. Reidel, 1979.
- [29] E. Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.
- [30] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *IEEE Softw.*, 18(2):22–29, 2001.
- [31] E. Carmel and P. Tjia. *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, 2005.
- [32] V. Casey and I. Richardson. Practical experience of virtual team software development. In *European Software Process Improvement*, 2004.
- [33] T. Chau, F. Maurer, and G. Melnik. Knowledge sharing: Agile methods vs. tayloristic methods. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 302–307. IEEE Computer Society, 2003.
- [34] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of haskell programs. In *Proceedings of the ACM Sigplan International Conference on Functional Programming (ICFP-00)*, volume 35.9 of *ACM Sigplan Notices*, pages 268–279, N.Y., sep 2000. ACM Press.
- [35] H.H. Clark and S.E. Brennan. Grounding in communication. In *In Resnick, L.B., Levine, J.M. and Teasley, S.D., Perspectives on social shared cognition*, pages 127–149, Washington, DC, USA, 1990. American Psychological Association.
- [36] M. Clifton and J. Dunlap. What is scrum? *codeproject*, 2003.
- [37] P. Coad, E. Lefebvre, and J. De Luca. *Java Modeling in Color with UML*. Prentice Hall, 1999.
- [38] A. Cockburn. *Writing effective use cases, The crystal collection for software professionals*. Addison Wesley, 2000.
- [39] A. Cockburn. *Agile software development*. Addison-Wesley, 2002.
- [40] A. Cockburn and J. Highsmith. Agile software development: The people factor. *Computer*, 34(11):131–133, 2001.
- [41] D Cohen, M. Lindvall, and P. Costa. An introduction to agile methods. *Advances in Computers*, 62:2–67, 2004.
- [42] B. Collins-Sussman, B.W. Fitzpatrick, and C.M. Pilato. *Version Control with Subversion*. O’Reilly & Associates, Sebastopol, California, June 2004.

- [43] E.Ó. Conchúir, H. Holmström Olsson, P.J. Ågerfalk, and B. Fitzgerald. Exploring the assumed benefits of global software development. *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 159–168, Oct. 2006.
- [44] E.Ó. Conchúir, H. Holmström Olsson, P.J. Ågerfalk, and B. Fitzgerald. Global software development: Never mind the problems are there really any benefits? *In Proceedings of the ICGSE International Conference on Global Software Engineering*., 2006.
- [45] J.M. Corbin and A.C. Strauss. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage, 2007.
- [46] D. Damian and D. Moitra. Guest editors' introduction: Global software development: How far have we come? *Software, IEEE*, 23(5):17–19, Sept.-Oct. 2006.
- [47] D.E. Damian and D. Zowghi. The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 319–328, 2002.
- [48] T. DeMarco. *Slack, Getting Past Burnout, BusyWork, and the Myth of Total Efficiency*. Broadway Books, 2002.
- [49] G. DeSanctis and P. Monge. Introduction to the special issue: Communication processes for virtual organizations. *Organization Science*, 10(6):693–703, 1999.
- [50] C. Ebert and P. De Neve. Surviving global software development. *Software, IEEE*, 18(2):62–69, Mar/Apr 2001.
- [51] C. Ebert, C.H. Parro, R. Suttels, and H. Kolarczyk. Improving validation activities in a global software development. *In ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 545–554, Washington, DC, USA, 2001. IEEE Computer Society.
- [52] J.A. Espinosa and E. Carmel. The effect of time separation on coordination costs in global software teams: a dyad model. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10 pp.–, Jan. 2004.
- [53] S. Faraj and L. Sproull. Coordinating expertise in software development teams. *Manage. Sci.*, 46(12):1554–1568, 2000.
- [54] S.I. Feldman. Make- a program for maintaining computer programs. *Software - Practice and Experience*, 9:255–265, 1979.
- [55] A. Fontana and J.H. Frey. The interview: From neutral stance to political involvement. *In N.K. Denzin and Y.S. Lincoln, editors, The Sage Handbook of Qualitative Research*, pages 695–727. Sage Publications, 2005.
- [56] M. Fowler. *The new methodology*. 2000.

-
- [57] M. Fowler. Using agile software process with offshore development. *martin-fowler.com*, july 2004.
- [58] M. Fowler. Continuous integration. 2006.
- [59] R.L. Freeman. *Telecommunication System Engineering*. John Wiley & Sons, Inc., 2004.
- [60] T. L. Friedman. *The World Is Flat: A Brief History of the Twenty-First Century*. Farrar, Straus and Giroux, 2005.
- [61] S.J. Galler, B. Peischl, and F. Wotawa. Automatic test generation tools for java based on design-by-contract: A survey. SNA TECHNICAL REPORT SNA-TR-2007-1P4, Softnet Austria Competence Network, september 1997.
- [62] R.L. Glass. Extreme programming: The good, the bad, and the bottom line. *IEEE Softw.*, 18(6):112, 2001.
- [63] C. Greenhalgh and S. Benford. Massive: a collaborative virtual environment for teleconferencing. *ACM Trans. Comput.-Hum. Interact.*, 2(3):239–261, 1995.
- [64] R.E. Grinter, J.D. Herbsleb, and D.E. Perry. The geography of coordination: dealing with distance in r&d work. In *GROUP '99: Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 306–315, New York, NY, USA, 1999. ACM.
- [65] D. Grune. Concurrent versions system, a method for independent cooperation. Technical report, IR 113, Vrije Universiteit, 1986.
- [66] D.-C. Gumm. Mutual dependency of distribution, benefits and causes: An empirical study. *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 113–124, Aug. 2007.
- [67] E.T. Hall. *Beyond culture*. Anchor Press, 1976.
- [68] B. Hammersley. *Content Syndication with RSS: Sharing Headlines and Information Using XML*. O'Reilly, 2003.
- [69] T. Hataria. The confounding world of process methodologies. In *Proc. CCEC 2006 Symposium*, 2006.
- [70] James D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE '07: 2007 Future of Software Engineering*, pages 188–198, Washington, DC, USA, 2007. IEEE Computer Society.
- [71] J.D. Herbsleb and R.E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 85–95, 1999.

- [72] J.D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, June 2003.
- [73] J.D. Herbsleb, A. Mockus, T.A. Finholt, and R.E. Grinter. Distance, dependencies, and delay in a global collaboration. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 319–328, New York, NY, USA, 2000. ACM.
- [74] J.D. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16–20, Mar/Apr 2001.
- [75] J.D. Herbsleb, D.J. Paulish, and M. Bass. Global software development at siemens: experience from nine projects. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 524–533, New York, NY, USA, 2005. ACM.
- [76] G. Hertel, S. Geister, and U. Konradt. Managing virtual teams: A review of current empirical research. *Human Resource Management Review*, 15(1):69–95, March 2005.
- [77] J. Highsmith. Messy, exiting, and anxiety-ridden: Adaptive software development. *American Programmer*, 10(1), 1997.
- [78] J. Highsmith and R.K. Wysocki. How agile are organizations today? *Cutter Consortium*, 7(12), dec 2006.
- [79] J.A. Highsmith. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Addison Wesley, 2000.
- [80] J.A. Highsmith. *Agile Software Development Ecosystems*. Addison Wesley, 2002.
- [81] G. Hofstede. Cultural constraints in management theories. *Academy of Management Executive*, 7(1):81–94, 1993.
- [82] B. Holmström Olsson, B. Fitzgerald, P.J. Ågerfalk, and E.Ó. Conchúir. Agile practices reduce distance in global software development. *Information Systems Development*, 23(3):7–18, 2006.
- [83] H. Holmström Olsson, E.Ó. Conchúir, P.J. Ågerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 3–11, Oct. 2006.
- [84] J.C. Jacobs and J.H. van Moll. *Effects of Virtual Product Development on Product Quality and their Influencing Factors*. PhD, Eindhoven University of Technology, 2007.

-
- [85] I. Jacobson, G. Booch, and J. Rumbaugh. The unified process. *IEEE Softw.*, 16(3):96–102, 1999.
- [86] K.H. Judy and I. Krumins-Beens. Great scrums need great product owners: Unbounded collaboration and collective product ownership. In *Proceedings of the 41st Hawaii International Conference on System Sciences*, page 462. IEEE Computer Society, 2008.
- [87] J. Kalermo and J. Rissanen. *Agile software development in theory and practice*. Master thesis, University of jyvaskylä, 2002.
- [88] B. Kaplan and J.A. Maxwell. Qualitative research methods for evaluating computer information systems. In J.G. Anderson and C.E. Aydin, editors, *Evaluating the Organizational Impact of Healthcare Information Systems*, Health Informatics, pages 30–55. Springer, 2005.
- [89] P. Keil, D.J. Paulish, and R.S. Sangwan. Cost estimation for global software development. In *EDSER '06: Proceedings of the 2006 international workshop on Economics driven software engineering research*, pages 7–10, New York, NY, USA, 2006. ACM.
- [90] L. Kiel. Experiences in distributed development: a case study. In *The International Workshop on Global Software Development, ICSE*, pages 44–47, 2003.
- [91] M. Kircher, P. Jain, A. Corsaro, and D. Levine. Distributed extreme programming. In *Proceedings of the International Conference on eXtreme Programming and Flexible Processes in Software Engineering*, pages 20–23, May 2001.
- [92] M. Korkala and P. Abrahamsson. Communication in distributed agile development: A case study. In *EUROMICRO '07: Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 203–210, Washington, DC, USA, 2007. IEEE Computer Society.
- [93] C. Larman. *Agile & Iterative Development - A Manager's Guide*. Addison-Wesley, 2004.
- [94] L. Layman, L. Williams, D. Damian, and H. Bures. Essential communication practices for extreme programming in a global software development team. *Information & Software Technology*, 48(9):781–794, 2006.
- [95] N.G. Leveson. Software engineering: Stretching the limits of complexity. *Communications of the ACM*, 40(2):129–131, February 1997.
- [96] B. Lings, B. Lundell, P.J. Ågerfalk, and B. Fitzgerald. A reference model for successful distributed development of software systems. *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 130–139, Aug. 2007.
- [97] J. Lipnack and J. Stamps. *Virtual Teams: People Working Across Boundaries with Technology, Second Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

- [98] D. Marinov and S. Khurshid. Testera: A novel framework for automated testing of java programs. In *Proceedings of the 16th IEEE Conference on Automated Software Engineering (ASE 2001)*. IEEE, nov 2001.
- [99] P.E. McMahon. Distributed development: Insights, challenges, and solutions. *CrossTalk - The Journal of Defense Software Engineering*, November 2001.
- [100] A. Mehrabian. *Nonverbal communication*. Aldine & Atherton, 1972.
- [101] A. Metiu and B. Kogut. Distributed knowledge and the global organization of software development. *Oxford Review of Economic Policy*, 17(02):248–264, 2002.
- [102] M.B. Miles and M. Huberman. *Qualitative Data Analysis: An expanded sourcebook*. Sage, 1994.
- [103] A. Mockus and J. Herbsleb. Challenges of global software development. *Software Metrics, IEEE International Symposium on*, 0:182, 2001.
- [104] A. Mockus and D.M. Weiss. Globalization by chunking: a quantitative approach. *Software, IEEE*, 18(2):30–37, Mar/Apr 2001.
- [105] N.B. Moe and D. Smite. Understanding lacking trust in global software teams: A multi-case study. In J. Münch and P. Abrahamsson, editors, *Product-Focused Software Process Improvement, 8th International Conference, PROFES 2007, Riga, Latvia, July 2-4, 2007, Proceedings*, volume 4589 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2007.
- [106] G.M. Olson and J.S. Olson. Distance matters. *Human-Computer Interaction*, 15(2/3):139–178., Sep 2000.
- [107] N. Oza, T. Hall, A. Rainer, and S. Grey. Critical factors in software outsourcing: a pilot study. In *WISER '04: Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*, pages 67–71, New York, NY, USA, 2004. ACM.
- [108] M. Paasivaara and C. Lassenius. Using iterative and incremental processes in global software development. *IEE Seminar Digests*, 2004(912):42–47, 2004.
- [109] M. Paasivaara and C. Lassenius. Could global software development benefit from agile methods? *Global Software Engineering, 2006. ICGSE '06. International Conference on*, pages 109–113, Oct. 2006.
- [110] S.R. Palmer and M. Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 2001.
- [111] K. Parvathanathan, A. Chakrabarti, P.P. Patil, S. Sen, N. Sharma, and Y. Johng. *Global development and delivery in practice experiences of the IBM Rational India Lab*. IBM, International Technical Support Organization, 2007.

-
- [112] M.Q. Patton. *Qualitative Evaluation and Research Methods*. Sage Publications, 1990.
- [113] E.R. Pedersen, K McCall, T.P Moran, and F.G. Halasz. Tivoli: an electronic white-board for informal workgroup meetings. In *CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, pages 391–398, New York, NY, USA, 1993. ACM.
- [114] C. Peersman, S. Cvetkovic, P. Griffiths, and H. Spear. The global system for mobile communications short message service. *Personal Communications, IEEE*, 7(3):15–23, Jun 2000.
- [115] D.E. Perry. Version control in the inscape environment. In *Proceedings of the 9th International Conference on Software Engineering*, pages 142–149, March 1987.
- [116] K. Pohl. The three dimensions of requirements engineering: a framework and its applications. *Information Systems*, 19(3):243–258, 1994.
- [117] C. J. Poole. Distributed product development using extreme programming. In *Extreme Programming and Agile Processes in Software Engineering, 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004, Proceedings*, volume 3092 of *Lecture Notes in Computer Science*, pages 60–67. Springer, 2004.
- [118] R.J. Potter. Electronic Mail. *Science*, 195(4283):1160–1164, 1977.
- [119] I.S.W.B. Prasetya, T.E.J. Vos, and A. Baars. Trace-based reflexive testing of oo programs with t2. *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 151–160, April 2008.
- [120] R. Prikladnicki, J.L.N. Audy, D. Damian, and T.C. de Oliveira. Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring. *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 262–274, Aug. 2007.
- [121] R. Prikladnicki, R. Evaristo, D. Damian, and J.L.N. Audy. Conducting qualitative research in an international and distributed research team: Challenges and lessons learned. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, page 442. IEEE Computer Society, 2008.
- [122] K.F. Punch. *Introduction to social research: Quantitative and qualitative approaches*. Sage, 1998.
- [123] J. Pyysiäinen. Building trust in global inter-organizational software development projects: Problems and practices. In *The International Workshop on Global Software Development, ICSE*, pages 69–74, 2003.
- [124] F. Rafii. How important is physical collocation to product development success? *Business Horizons*, 38(1):78–84, 1995.

- [125] B. Ramesh, L. Cao, K. Mohan, and P. Xu. Can distributed software development be agile? *Commun. ACM*, 49(10):41–46, 2006.
- [126] S.A. Rice. Hypotheses and verifications in clifford r. shaw’s studies of juvenile delinquency. In *Methods in Social Science*, pages 549–565. University of Chicago Press, 1931.
- [127] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
- [128] W. Royce. Managing the development of large software systems. In *Proc. IEEE Wescon*, pages 1–9, August 1970.
- [129] O. Salo and P. Abrahamsson. Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. *Software, IET*, 2(1):58–64, February 2008.
- [130] R. Sangwan, M. Bass, N. Mullick, D.J. Paulish, and J. Kazmeier. *Global Software Development Handbook*. Auerbach Publications, 2007.
- [131] S. Sarker and S. Sahay. Implications of space and time for distributed work: an interpretive study of us-norwegian systems development teams. *Eur. J. Inf. Syst.*, 13(1):3–20, 2004.
- [132] K. Schwaber. Scrum development process. In *OOPSLA’95 Workshop on Business Object Design and Implementation*, 1995.
- [133] K. Schwaber and Beedle M. *Agile Software Development with Scrum*. Alan R. Apt, first edition, 2001.
- [134] H.J. Shatz and A.J. Venables. The geography of international investment. Policy Research Working Paper Series 2338, The World Bank, May 2000.
- [135] M. Simons. Internationally agile. *InformIT*, march 2002.
- [136] S.J. Taylor and R. Bogdan. *Introduction to Qualitative Research Methods*. John Wiley & Sons, 1984.
- [137] Shine Technologies. Agile methodologies survey results. 2003.
- [138] W.F. Tichy. Tools for software configuration management. In *Proceedings of the International Workshop on Software Version and Configuration Control*, pages 1–20, January 1988.
- [139] J. van Moll, J. Jacobs, R. Kusters, and J. Trienekens. Defect detection oriented life-cycle modeling in complex product development. *Information and Software Technology*, 46(10):665–675, august 2004.

-
- [140] M.-A. Vanzin, M.B. Ribeiro, R. Prikladnicki, I. Ceccato, and D. Antunes. Global software processes definition in a distributed environment. *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA*, pages 57–65, April 2005.
- [141] L. Williams. A survey of agile development methodologies. 2004.
- [142] C. Wohlin, M. Höst, and K Henningsson. Empirical research methods in software engineering. In Reidar Conradi and Alf Inge Wang, editors, *Empirical Methods and Studies in Software Engineering, Experiences from ESERNET*, volume 2765 of *Lecture Notes in Computer Science*, pages 7–23. Springer, 2003.
- [143] R.K. Yin. *Case study research : design and methods*. SAGE Publications, Thousand Oaks, California, 1994.
- [144] D.E. Zand. *The leadership triad : knowledge, trust, and power*. Oxford University Press, New York, 1997.

Appendix A

Interview Structure

Main Goal:

To elicit the main requirements of an ICDE

Sub Goal:

To elicit:

- How cooperation in both co-located and distributed teams takes place
- What exactly is more difficult in a distributed team in comparison with a co-located team with respect to cooperation
- How these problems are currently dealt with and how satisfying this is

Guiding questions:

1. Eliciting context information:

- a) What kind of development does your company do (product versus project development)?
- b) What kind of development teams does your company have? Examples of variables to describe the development teams are:
 - i. The number of teams
 - ii. The geographical location
 - iii. The size
 - iv. The social cultural background
 - v. Do different teams cooperate in distributed fashion or are the teams actually distributed themselves?
- c) Describe an *"average"* work day of yourself
- d) How well is the cooperation:
 - i. Within the co-located teams
 - ii. Within the distributed teams

e) How does this correspond with you initial expectations?

2. Eliciting common user stories or scenarios with respect to collaborative development

Here we attempt to asses in what way several activities carried out in the development process can be considered collaborative activities. The activities we consider are the following:

- a) Planning
- b) Requirement clarification
- c) Design
- d) Quality assurance
 - i. Testing
 - ii. Code review
- e) Construction
- f) Integration
- g) Maintenance

The leading questions we ask for each of these activities are:

- a) How is this activity performed at your company?
- b) How is information regarding it being documented and communicated? (How is knowledge regarding the activity being shared?)
- c) When carrying out this activity with whom do you cooperate and in what way?
- d) How does being physically distributed from you colleagues make this activity harder?
- e) How do you attempt to solve these issues?
- f) What are the limitations of these solutions?

3. Eliciting the negative impact of working in a distributed setting on informal, non work-related, communication

- a) To what degree do you communicate in an informal non work-related fashion with your co-located colleagues
- b) To what degree do you communicate in an informal non work-related fashion with your colleagues who are located elsewhere
- c) (If there is a difference between a. and b.) Describe the reason for this difference
- d) How do you try to prevent this difference?
- e) Do you have ideas of yourself on how this situation could be improved upon? (Both technological and non-technological)

4. **Eliciting the extend to which technological support is currently integrated and ideas on how to improve this**

Here we split the discussion into three categories:

- a) Integration between communication related technologies
- b) Integration between software development related technologies
- c) Integration between communication related and software development related technologies

The leading questions we ask for each of these are:

- a) What tooling do you use for this purpose?
- b) What goal does using each of these technologies have?
- c) Describe the integration between different technologies of this category
- d) Propose improvements or extensions to integrating the supporting technologies

Appendix B

Requirements of an ICDE

Main

Allow for efficient collaboration among the people of the development team to enable the creation of high quality software.

Sub-Requirements:

RC_1 : Incorporate the knowledge about the project into the development environment

RC_2 : Facilitate inter-personal contact

RC_3 : Derive information from the system

Coupling requirements to requirement categories:

RC_1 : Integration of R_2 and R_3 into the development environment

RC_2 : Integration of R_1 and R_3 into the development environment

RC_3 : Integration of R_4 and R_5 into the development environment

RC_1 : Incorporate the knowledge about the project into the development environment

- *Context linking (annotate work items)*
 - Linking of work items to related actors
 - ★ example: The initial reporter of an issue linked to that issue
 - ★ example: The author of a piece of code to that piece of code
 - Link work items to a related discussion
 - Link related work items together
 - ★ example: A requirement linked to the corresponding test cases, use cases and issues.
 - ★ example: A commit on a branch linked to a resolved work item (and possibly also to an entry in the activity-list in the HRM-system)
- *Provide a "dashboard" with the current status of the project*
 - Functional/Integration/regression/metrics test status

- Organizational
 - ★ Indicator for percentage of planned tasks which are completed
 - ★ Divergence with respect to planning
 - ★ example: SCRUM board / burn down chart
- Overview of configuration machines (both for testing and maintenance) and if they are patched sufficiently

Clarification: A dashboard regards an overview with red/orange/yellow and green squares indicating the status

- *Notification of certain events regarding the system*
 - example: failed build
 - ★ author of a failed test
 - ★ author of a failed use case
 - Various communication media
 - ★ RSS, IM, mail, IRC, Twitter
 - ★ The dashboard

RC₂: Facilitate inter-personal contact *Technological*

- *Make it easy to change the richness of communication with a specific individual*
 - This regards switching between communication media
 - This includes switching from no contact at all to an active conversation
- *Status of people should be clear, up-to-date and accurate (connected with all communication media)*
 - example: Phone occupied
 - example: Agenda
 - example: Camera on room (overview)
Just to see people sit behind their computers and work
 - example: Status based on the view people have in their development environment
- *The following communication media are desirable:*
 - Text
 - Audio
 - Video
 - Context sharing (screen sharing)
Seeing the others context is more important than being able to directly influence it

-
- Visualization
 - ★ example: Drawing something on a board

Non-technological approaches

- *Distributed meetings Explicit process (when large-scale)*
 - Planning poker
 - Clear agenda
 - With discussion-leader
 - ★ Being able to put questions on hold
 - ★ All sorts of other administration rights

Uniform discussion environment

- Environment in which the communication with all actors in the meeting is uniform
 - ★ *Pro*: prevent co-located people from having a local discussion which other actors are not able to follow
 - ★ *Pro*: Actors feel more equally connected to all participants in the discussion
 - ★ *Con*: This approach might seem very unnatural when you are talking to a co-located colleague in the next room
 - ★ *Con*: The advantages with respect to having a discussion when being collocated are lost between the actors that could potentially share the same room

- *Improving informal communication*

Why?

- To lower the threshold to ask for help
 - ★ Because a better view of your colleagues capabilities
 - ★ To know how to approach your colleagues (socially)
 - ★ When asking for help someone that knows you socially will be more willing to help you
- To improve the overall working experience of the developers

How?

- Work a certain period of time per week (half a day) with a certain remote colleague (via webcam)
- Physically visiting the remote site for a certain period of time
- Making use of "informal" communication media such as: Weblogs/newsletter/Twitter etc.

B. REQUIREMENTS OF AN ICDE

- ★ Colleagues will know something about each other
- ★ Colleagues will have a topic to start a random conversation
- Exchange of ambassadors between team which will act as a proxy between their new and original site.
- Performing the first part of the project same site
 - ★ Colleagues will get to know each other better
 - ★ Doing the initial work together leads to feeling of common ownership which in turn increased the teamness
- Periodical presentations of separate teams (which collaborate) will help to keep clear what everyone is doing
 - ★ People will feel more like a team
 - ★ Contacting people you have seen before is easier
- Asking something via phone(audio) as apposed to chat could help to initiate informal contact
- Calling remote contacts without a specific topic to discuss once a week will ease asking each other questions, updating each other and informal communication
 - ★ Unconditional contact appointments
- Having a specific remote contact person (virtual neighbor or 'buddy')
 - ★ Have regular contact
 - ★ Helps to keep the distributed teams connected

RC₃: Derive information from the system

- *Connecting the continuous build server to the quality assurance to produce continuous quality assurance*
 - Functional testing
 - Code-metrics (static analysis)
 - ★ Objective assessment of code
 - Regression testing
- *An environment for multiple versions of the software*
 - A full configuration of a development machine for each version (including the state of the IDE)
 - ★ Virtual machine image
 - ★ So, patched just like the machine(s) on which this version was last developed
 - For each version a collection of configurations (virtual machines) on which that specific version must work to be able to automatically test on.
 - ★ So, a configuration with the different versions of the software the system works with (like different OS, different version of an OS, different VM, different database architectures etc.)