

Darja Šmite
Nils Brede Moe
Pär J. Ågerfalk (Eds.)

Agility Across Time and Space

Implementing Agile Methods
in Global Software Projects



Springer

Agility Across Time and Space

Darja Šmite • Nils Brede Moe • Pär J. Ågerfalk
Editors

Agility Across Time and Space

Implementing Agile Methods
in Global Software Projects

 Springer

Editors

Darja Šmite
School of Engineering
Blekinge Institute of Technology
372 25 Ronneby
Sweden
Darja.Smite@mac.com

Nils Brede Moe
Dept. Information
& Communication Technology (ICT)
SINTEF
7465 Trondheim
Norway
Nils.b.moe@sintef.no

Pär J. Ågerfalk
Dept. Information Sciences
Uppsala University
Box 513
751 20 Uppsala
Sweden
Par.Agerfalk@im.uu.se

ISBN 978-3-642-12441-9
DOI 10.1007/978-3-642-12442-6
Springer Heidelberg Dordrecht London New York

e-ISBN 978-3-642-12442-6

Library of Congress Control Number: 2010927809

ACM Computing Classification (1998): D.2, K.6

© Springer-Verlag Berlin Heidelberg 2010

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Cover design: KünkelLopka, Heidelberg

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Foreword

Drawing from earlier definitions from Jim Highsmith or Steve Adolph and the OODA loop, I like to define agility as “the ability of an organization to react to change in its environment faster than the rate of these changes.” This definition uses the ultimate purpose or function of being agile for a business, rather than defining agility by a labeled set of practices (e.g., you’re agile when you do XP, Lean, or Scrum) or by a set of properties defined in opposition to another set (the agile manifesto approach).

An analogy could be the definition of a road. Would you define a road as something made of crushed rocks and tar, or define it as a surface that is black rather than white, flat rather than undulated, and with painted lines rather than monochrome? Or as a component of a transportation system, allowing people and goods to be moved on the ground surface from point A to point B? And let the properties or components be derived from this, allowing some novel approach in road design.

It is quite possible to adopt a labeled set of agile practices, or a set of practices that perfectly conform to the agile manifesto and not become agile. You then “do Agile” but are not agile.

Agile software development methods do succeed in contexts which are identical or very similar to the contexts in which they have been created. As these contexts—the “agile sweet spot”—are very frequent in software development, representing more than 50% of all software being developed, this may have led sometimes their proponents to a certain complacency: thinking that the method has universal value, that it represents some ultimate recipe, the holy grail.

Agile methods may fail in various ways when they are applied “out of the box”, i.e., with no or little adaptation, in contexts that are very far, at least on one dimension, from the context in which they have been originally created. Rather than an analysis of the root cause, this usually triggers screams of “you must have not done it right” by its proponents. And this again leads to discussion of “purity”, “scrum-butts”, etc.

Agile methods can be stretched with variable success outside of the context in which they have been created; for example, scaling them up to larger projects, or across distributed teams. In my experience, the contextual factors that have the greatest risks of derailing agile projects are:

- size
- large systems with a lack of architectural focus
- software development not driven by customer demand
- lack of support from surrounding stakeholders, traditional governance
- novice team
- very high constraint on some quality attribute (safety-critical system, real-time constraints)

As noted by many authors in the last few years, we cannot just rely on acts of faith by eloquent process gurus to help us define the adequate process, or set of practices outside of the agile sweet spot. Cold-headed, impartial investigation is required. Such research is generally not very easy to conduct; it is often qualitative, rather than quantitative, it draws more from social sciences than computer science, not easy to publish, not easy to carve down to masters' thesis bite size.

This is the reason why I welcome this volume on *agility across time and space*. Looking at how agile practices performed once stretched outside of the agile sweet spot, for large projects and distributed projects, the non-trivial ones. The researchers and practitioners who collectively wrote this volume have been examining without prejudice what works and what does not, and trying to get at the root cause, giving us another and better perspective on this fascinating wave: the agile software development movement. They confront some of the factors I mentioned earlier: size, distribution, role of architecture, culture.

Because all things considered, our stakeholders do not care whether you did or not your daily stand-up meetings, whether pairing was followed religiously, how many columns in your kanban, or if you played poker for estimations. They only care about quality software hitting the market as fast as we possibly can. The software developer should only be concerned by what will allow her to achieve this in her specific context. And in a turbulent environment, can the organization react to change in its environment faster than the rate of these changes?

Vancouver, BC, Canada

Philippe Kruchten

Preface

Despite the progress in the field of software engineering, software projects are still being late, are over budget, and do not deliver the expected quality. Two major trends have emerged in response to these: global sourcing and the application of agile methods. The new paradigms soon became anecdotally popular for their benefits of cheaper and faster development of high quality software. Many companies recently started to look into merging these two promising approaches into one strategy.

Globally Distributed Development

Global sourcing promises organizations the benefits of reaching mobility in resources, obtaining extra knowledge through deploying the most talented people around the world, accelerating time-to-market, increasing operational efficiency, improving quality, expanding through acquisitions, reaching proximity to market and many more. However, these benefits are neither clear-cut nor can their realization be taken for granted, as the literature may lead one to believe [1]. In fact, there are many challenges related to communication, coordination and control when developing software with global software teams [2].

Agile Development

Agile development has recently attracted huge interest from software industry [3]. It is being recognized for its potential to improve communication and, as a result, reduce coordination and control overhead in software projects. Methods for agile software development constitute a set of practices for software development that have been created by experienced practitioners [4]. The “agile manifesto” was published in 2001 by the key people behind the early agile development methods. The manifesto states that agile development should focus on four core values [5]:

- Individuals and interactions over processes and tools,

- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation,
- Responding to change over following a plan.

Agile methods can be seen as a reaction to plan-based or traditional methods, which emphasize “a rationalized, engineering-based approach” [6] in which it is claimed that problems are fully specifiable and that optimal and predictable solutions exist for every problem. The “traditionalists” are said to advocate extensive upfront planning, codified processes, and rigorous reuse to make development an efficient and predictable activity [7]. By contrast, agile processes address the challenges of the increasingly complex nature of software development by relying on people and their creativity rather than on formalized processes [6]. The goal of optimization is being replaced by those of flexibility and responsiveness [8]. Ericksson et al. [9] define agility as follows: *agility means to strip away as much of the heaviness, commonly associated with the traditional software-development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated project deadlines and the like.* (p. 89)

The Role of Agility in Distributed Development

Global software development has matured considerably since its inception and has become an integral part of the information technology landscape. Now, rather than deciding whether or not to get involved in global sourcing, many companies are facing decisions about whether or not to apply agile methods in their distributed projects. These companies are often motivated by the opportunities of solving the coordination and communication difficulties [4] associated with global software development.

Empirical evidence from case studies conducted by Paasivaara and Lassenius [10], and Holmström, Fitzgerald et al. [11] show successful implementation of agile values and principles in different globally distributed projects. This motivates assessing the viability of agile practices for distributed software development teams. The interest in becoming agile and distributed is also illustrated by the increasing number of research publications and seminars devoted to the topic.

Implementing Agility Across Time and Space

Despite the increased attention, merging the two strategies is no easy task due to significant differences in fundamental principles of agile and distributed development approaches. In particular, while agile principles prescribe close interaction and collocation, the very nature of distributed software development does not support these prerequisites. Taylor, Greer et al. [12] claim that distributed agile software development suffers substantial difficulties because of its complex development environment and there is little empirical evidence describing actual development experiences. The lack of clear understanding of who, what, when, why and how in agile

distributed development motivated us to collect experiences from various companies that had started, and also benefitted from, becoming agile and distributed.

Aims of the Book

The idea to write a book on agile and distributed software development gradually evolved as the critical mass of questions related to merging seemingly incompatible approaches emerged. The questions that the authors aimed to answer with this book include:

- What shall companies expect from merging agile and distributed strategies?
- What are the stumbling blocks that prevent companies from reaching the agile benefits in distributed environment, and how to recognize unfeasible strategies and unfavorable circumstances?
- What helps managers cope with the challenges of implementing agile approaches in distributed software development projects?
- How can distributed teams survive the decisions taken by the management and become efficient through the application of agile approaches?

Book Overview

This book consists of five parts.

1. In the **Motivation** part the editors introduce the fundamentals of agile distributed software development and explain the rationale behind the application of agile practices in globally distributed software projects.

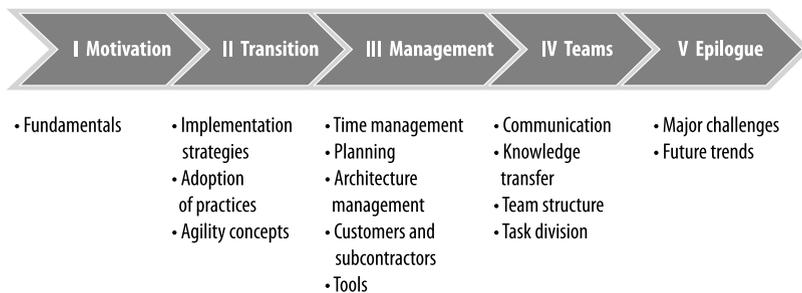


Fig. 1 Book layout

2. The second part of the book is called **Transition**. Here we have gathered seven chapters that discuss the transition to being agile and distributed. The chapters describe implementation strategies, adoption of particular agile practices for distributed projects, and general concepts of agility.

3. The third part of the book, **Management**, focuses on managerial aspects and decisions in agile distributed software projects. Practical implications for project planning, time management, customer and sub-contractor interaction, tool support and architecture-centric development are presented in eight chapters.
4. The fourth part is devoted to agile and distributed **Teams**. Here we have collected six chapters that provide in-depth hands-on advice for the team members and their managers. Topics discussed include agile distributed team configuration, effective communication and knowledge transfer, the role of architecture in task division, and allocation of roles and responsibilities.
5. finally, in the **Epilogue** we summarize the contributions of the different chapters and present results from a Delphi-inspired study that highlights the major areas of concern and future trends for research and practice in agile distributed development.

Most of the chapters in this book offer practical advice based on experiences obtained in and from the industry. These experiences are collected through personal observations of practitioners, empirical research in particular studied contexts or extensive continuous observations gained from various sources.

Target Audience

This book is primarily targeted at practitioners (managers and team members) involved in globally distributed software projects - those who are practicing agile methods and those who are not. We believe that it will serve as a useful source of practical advice, which are based on the real life examples of application of agile practices in distributed development, and will hopefully motivate companies to try improving their sourcing strategies by adopting best practices and benefits that agile promises.

Many book chapters are based on the sound empirical research and identify gaps and commonalities in the existing state-of-the-art and state-of-the-practice. We thus believe that our book can be also of relevance and interest for the academic audience, in particular, researchers working in the field, as well as lecturers and students of global agile software development.

References

1. Ó. Conchúir, E., Ågerfalk, P. J., Fitzgerald, B., & Holmström Olsson, H. (2009). Global software development: Where are the benefits?. *Communications of the ACM*, 52(8), 127–131.
2. Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., & Ó. Conchuir, E. (2005). A framework for considering opportunities and threats in distributed software development. In *Proceedings of the international workshop on distributed software development (DiSD)* (pp. 47–61). Vienna: Austrian Computer Society.
3. Dybå, T., & Dingsøy, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10), 833–859.

4. Ågerfalk, P. J., & Fitzgerald, B. (2006). Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM*, 49(10), 26–34.
5. Beck, K., et al. (2001). Agile manifesto. Available online. <http://agilemanifesto.org>. Cited 15 Feb 2010.
6. Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 72–78.
7. Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer*, 35(1), 64–69.
8. Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies—The traditional goal of optimization and control is making way for learning and innovation. *Communications of the ACM*, 50(3), 79–83.
9. Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management*, 16(4), 88–100.
10. Paasivaara, M., & Lassenius, C. (2006). Could global software development benefit from agile methods? In *International conference on global software engineering* (pp. 109–113).
11. Holmström, H., Fitzgerald, B., Ågerfalk, P. J., & Ó. Conchuir, E. (2006). Agile practices reduce distance in global software development. *Information Systems and Management*, 23(3), 7–18.
12. Taylor, P. S., Greer, D., Sage, P., Coleman, G., McDaid, K., & Keenan, F. (2006). Do agile GSD experience reports help the practitioner? In *Proceedings of the 2006 international workshop on global software development of ACM* (pp. 87–93).

*Darja Šmite
Nils Brede Moe
Pär J. Ågerfalk*

Acknowledgements

We are thankful to all the authors for their valuable contributions and effort in creation of this book. We are also thankful to the external reviewers for their timely responses and valuable feedback. Springer Computer Science Editorial and especially Ralf Gerstner deserves a special gratitude for initiating the creation of this book and supporting our ideas on its way. A special thanks goes to Likoebe M. Maruping for the book title idea and to Claes Wohlin who has brought us together. Last but not the least we are thankful to our families and friends for their immeasurable support.

This book has been supported by

- the Software Engineering Research Lab in the School of Computing, Blekinge Institute of Technology,
- the Research Project “Agile”, funded by the Research Council of Norway under grant 179851/I40, and
- the Department of Informatics and Media, Uppsala University

Contents

Part I Motivation

1	Fundamentals of Agile Distributed Software Development	3
	Darja Šmite, Nils Brede Moe, and Pär J. Ågerfalk	
1.1	Introduction	3
1.1.1	Distributed Software Development	3
1.1.2	Agile Software Development	4
1.2	Merging Agility with Distribution	4
1.2.1	Potential Issues	5
1.2.2	All or Nothing versus À la carte	6
1.3	Current Practice	6
1.4	Conclusions	7
	References	7

Part II Transition

2	Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges	11
	Likoebe M. Maruping	
2.1	Introduction	11
2.2	Implementing XP Practices: Where Is an Organization to Start?	12
2.2.1	The Promise of XP	12
2.2.2	Understanding How Your Software Project Team Is Structured and Why It Matters	13
2.3	Case Overview	14
2.4	XP in Distributed Software Project Teams: Implementation Strategies and Pitfalls to Avoid	16
2.4.1	The Planning Game	16
2.4.2	Collective Ownership	18
2.4.3	Coding Standards	19
2.4.4	Use of a Metaphor	20

- 2.4.5 Simplicity of Design 21
- 2.4.6 Sustainable Pacing 22
- 2.4.7 Pair Programming 23
- 2.4.8 Continuous Integration and Unit Testing 24
- 2.4.9 Refactoring 25
- 2.4.10 Customer Involvement 26
- 2.4.11 Small Functional Releases 27
- 2.5 Conclusions 28
- References 29
- Further Reading 30

- 3 Transitioning from Distributed and Traditional to Distributed and Agile: An Experience Report 31**
- Daniel Wildt and Rafael Prikladnicki
- 3.1 Introduction 31
- 3.2 Case Overview 32
- 3.3 Transitioning to Agile in a Distributed Environment 34
 - 3.3.1 Don’t Tell What Agile Is and Be Successful 35
 - 3.3.2 A Fully Cultural Transition from Traditional to Agile Development 37
 - 3.3.3 Benefits of Using Agile Methods in Distributed Environment 40
- 3.4 Practical Recommendations 41
- 3.5 Conclusions 45
- References 45

- 4 Tailoring Agility: Promiscuous Pair Story Authoring and Value Calculation 47**
- Steve Tendon
- 4.1 Introduction 47
- 4.2 The Case 48
 - 4.2.1 Background 48
 - 4.2.2 Management Support and Sponsorship 50
 - 4.2.3 The Pilot Project 51
 - 4.2.4 The Journey of Implementing Agility 53
 - 4.2.5 The Final: Project Approval 61
- 4.3 Benefits from Implementing Agility over Traditional Approaches 62
 - 4.3.1 More Commonality 62
 - 4.3.2 Smaller Scope 63
 - 4.3.3 ROI Anticipation 63
 - 4.3.4 Smaller Country-Specific Dependencies 63
 - 4.3.5 Avoiding Waste Upfront 63
- 4.4 Why Agile Succeeded? 64
 - 4.4.1 Induction 64
 - 4.4.2 Co-location and Alternating On- and Off-Site Activities 65
 - 4.4.3 Promiscuous Pair Story Authoring 66

- 4.4.4 Economic Value of Story Points 67
- 4.5 Conclusions 68
- References 69
- 5 Scrum and Global Delivery: Pitfalls and Lessons Learned 71**
- Cristiano Sadun
- 5.1 Introduction 71
- 5.2 Cases Overview 72
- 5.2.1 Background 72
- 5.2.2 Project NOR1 74
- 5.2.3 Project NOR2 76
- 5.3 The Experiences 77
- 5.3.1 Signing Agreements 77
- 5.3.2 Establishing Remote Access 79
- 5.3.3 Overcoming Communication Barriers 80
- 5.3.4 Actively Managing Distributed Agile Projects 82
- 5.3.5 Dealing with Idle Time 84
- 5.3.6 Achieving Motivation and Peer Feeling 86
- 5.3.7 Adapting Governance and Steering 87
- 5.4 Conclusions 88
- References 89
- 6 Onshore and Offshore Outsourcing with Agility: Lessons Learned . . 91**
- Clifton Kussmaul
- 6.1 Introduction 91
- 6.2 Case Overview 92
- 6.2.1 Background 92
- 6.2.2 Project Organization 93
- 6.2.3 Introduction of Agility 95
- 6.2.4 Overview of Project Activities 95
- 6.2.5 Cross-border Relationship Dynamics 97
- 6.3 Lessons Learned 98
- 6.3.1 People 99
- 6.3.2 Processes 100
- 6.3.3 Coordination 102
- 6.4 Conclusions 104
- References 105
- Further Reading 105
- 7 Contribution of Agility to Successful Distributed Software Development 107**
- Saonee Sarker, Charles L. Munson, Suprateek Sarker, and Suranjan Chakraborty
- 7.1 Introduction 107
- 7.2 Distributed Project Success 108
- 7.3 Types of Agility 109

- 7.4 Study Background 109
- 7.5 Contribution of Agility to Distributed Project Success 112
- 7.6 Conclusions 114
- References 116

8 Preparing your Offshore Organization for Agility: Experiences in India 117

Jayakanth Srinivasan

- 8.1 Introduction 117
- 8.2 Distributed Agile Software Development in India 118
- 8.3 Experiences from AgileCo 119
 - 8.3.1 Case Overview 119
 - 8.3.2 Personnel Selection and Training 120
 - 8.3.3 Teaching and Mentoring 122
 - 8.3.4 Managing Customer Expectations 123
- 8.4 Experience from BankCo 124
 - 8.4.1 Case Overview 124
 - 8.4.2 Impact of Senior Leadership Vision 125
 - 8.4.3 Heterogeneous Process Environment 126
 - 8.4.4 Agile Coaching 127
- 8.5 Conclusions 127
- References 129

Part III Management

9 Improving Global Development Using Agile 133

Alberto Avritzer, Francois Bronsard, and Gilberto Matos

- 9.1 Introduction 133
- 9.2 The Projects 134
- 9.3 Deploying Agile Techniques in Global Projects 136
 - 9.3.1 Organizational Issues 136
 - 9.3.2 Communication Issues 137
 - 9.3.3 Process Issues 139
 - 9.3.4 Tools and Technical Issues 141
- 9.4 Improving Global Projects Using Agile Processes 143
- 9.5 Conclusions 147
- References 147

10 Turning Time from Enemy into an Ally Using the Pomodoro Technique 149

Xiaofeng Wang, Federico Gobbo, and Michael Lane

- 10.1 Introduction 149
- 10.2 Time Is an Enemy? 151
- 10.3 The Pomodoro Technique 152
 - 10.3.1 Pomodoro as Time-box 153
 - 10.3.2 Pomodoro as Unit of Effort 153

- 10.4 The Application of the Pomodoro Technique in Sourcesense
 - Milan Team 154
 - 10.4.1 Background of Sourcesense Milan Team 154
 - 10.4.2 The Development Process of Sourcesense Milan Team . . . 155
 - 10.4.3 Pomodoro as Time-box 156
 - 10.4.4 Pomodoro as a Unit of Effort 159
 - 10.4.5 Addressing Remote Collaboration with Teams That Do Not Employ the Pomodoro Technique 161
- 10.5 Turning Time into an Ally 161
 - 10.5.1 Shared Pomodoro 162
 - 10.5.2 Collective Breaks 162
 - 10.5.3 Estimation and Tracking 163
 - 10.5.4 One Pomodoro Rules All Sites? 163
- 10.6 Conclusions 164
 - References 165
- 11 MBTA: Management By Timeshifting Around 167**
 - Erran Carmel
 - 11.1 Management by Wandering and Flying Around 167
 - 11.2 Enter Timeshifting 168
 - 11.3 Conclusions 170
 - References 170
- 12 The Dilemma of High Level Planning in Distributed Agile Software Projects: An Action Research Study in a Danish Bank 171**
 - Per Svejvig and Ann-Dorte Fladkjær Nielsen
 - 12.1 Introduction 171
 - 12.2 Research Methodology 173
 - 12.2.1 Action Research 173
 - 12.2.2 Research Settings 173
 - 12.3 The Action Research Cycle 174
 - 12.3.1 Diagnosing the Problem and the Underlying Causes 174
 - 12.3.2 Action Planning 175
 - 12.3.3 Action Taking 175
 - 12.3.4 Evaluating and Learning 180
 - 12.4 Conclusions 181
 - 12.4.1 Applying a Holistic Approach to High Level Planning . . . 181
 - 12.4.2 Using Action Research to Software Process Improvement . 182
 - 12.4.3 Summary 182
 - References 182
- 13 Tools for Supporting Distributed Agile Project Planning 183**
 - Xin Wang, Frank Maurer, Robert Morgan, and Josyleuda Oliveira
 - 13.1 Introduction 183
 - 13.2 Distributed Planning Tool Requirements 185
 - 13.2.1 Agile Planning Requirements 186

- 13.2.2 Requirements for Collaborative Interactions 187
- 13.3 Tool Review 188
 - 13.3.1 Wikis 188
 - 13.3.2 Web Form-Based Applications 189
 - 13.3.3 Card-Based Planning Systems 190
 - 13.3.4 Plugin for Integrated Development Environment 190
 - 13.3.5 Synchronous Project Planning Tool 191
 - 13.3.6 Digital Tabletop-Based Agile Planning Tool 193
- 13.4 Tool Evaluation 193
- 13.5 Practical Advice 195
 - 13.5.1 Advice for Agile Planning Tool User 195
 - 13.5.2 Advice for Designers of Distributed Agile Planning Tools . 196
- 13.6 Conclusions 198
 - References 199
- 14 Combining Agile and Traditional: Customer Communication in Distributed Environment 201**
 Mikko Korkala, Minna Pikkariainen, and Kieran Conboy
 - 14.1 Introduction 201
 - 14.2 Customer Communication in Distributed Agile Development . . . 202
 - 14.2.1 Issues Hindering the Customer Communication in Distributed Agile Development 204
 - 14.3 Findings 205
 - 14.3.1 Case Context 205
 - 14.3.2 The Use of Agile Methodologies in the Case Project 207
 - 14.3.3 The Use of Customer Communication Media 208
 - 14.3.4 Identified Customer Communication Challenges 211
 - 14.4 Discussion and Lessons Learned 214
 - References 216
- 15 Coordination Between Global Agile Teams: From Process to Architecture 217**
 Jan Bosch and Petra Bosch-Sijtsema
 - 15.1 Introduction 217
 - 15.2 Large-Scale Software Development 220
 - 15.3 Case Study Companies 221
 - 15.3.1 Case Company GLOembed 221
 - 15.3.2 Case Company GLOtelcom 222
 - 15.3.3 Case Company GLOsoftware 222
 - 15.4 Coordination and Integration Inter-team Challenges 224
 - 15.4.1 Top-Down Approach Challenges 224
 - 15.4.2 Interaction Problems 225
 - 15.5 Coordination Through Architecture 226
 - 15.5.1 Road Mapping 227
 - 15.5.2 Requirements 228
 - 15.5.3 Architecture 229

- 15.5.4 Development 229
- 15.5.5 Integration or Composition 230
- 15.5.6 Architecture-Centric Software Engineering 231
- 15.6 Conclusions 232
- References 233
- 16 Considering Subcontractors in Distributed Scrum Teams 235**
 Jakub Rudzki, Imed Hammouda, Tuomas Mikkola, Karri Mustonen,
 and Tarja Systä
 - 16.1 Introduction 235
 - 16.1.1 Company Context 236
 - 16.1.2 Methodology 236
 - 16.1.3 Main Results 237
 - 16.2 Subcontractors in an SSP Company 238
 - 16.2.1 Why Subcontractors? 239
 - 16.2.2 Distributed Development Stakeholders 239
 - 16.2.3 Subcontractor Selection Process 240
 - 16.3 Subcontractors in Scrum Teams 242
 - 16.3.1 Scrum 242
 - 16.3.2 Communication 243
 - 16.3.3 Planning and Progress Tracking 244
 - 16.3.4 Code Sharing and Development Feedback 245
 - 16.3.5 Knowledge Sharing 246
 - 16.3.6 Team Spirit 246
 - 16.4 Subcontractors and Project Phases 247
 - 16.4.1 Preparation 247
 - 16.4.2 Development 248
 - 16.4.3 Release 251
 - 16.5 Conclusions 251
 - 16.5.1 Practical Implications 252
 - 16.5.2 Research Implications 252
 - 16.5.3 Summary 253
 - Appendix 253
 - References 255
 - Further Reading 255

Part IV Teams

- 17 Using Scrum Practices in GSD Projects 259**
 Maria Paasivaara and Casper Lassenius
 - 17.1 Introduction 259
 - 17.2 Research Methodology 260
 - 17.3 Distributed Daily Scrums 260
 - 17.3.1 Application of Daily Scrums to Distributed Projects 262
 - 17.3.2 Benefits of Daily Scrums 263

- 17.3.3 Challenges of Daily Scrums 263
- 17.4 Scrum-of-Scrums Meetings 264
 - 17.4.1 Application of Scrum-of-Scrums to Distributed Projects . . 265
 - 17.4.2 Benefits of Scrums-of-Scrums 265
 - 17.4.3 Challenges of Scrums-of-Scrums 266
- 17.5 Sprints 266
 - 17.5.1 Application of Sprints to Distributed Projects 267
 - 17.5.2 Benefits of Sprints 267
 - 17.5.3 Challenges of Sprints 268
- 17.6 Sprint Planning Meetings 268
 - 17.6.1 Application of Sprint Planning Meetings to Distributed
Projects 268
 - 17.6.2 Benefits of Sprint Planning Meetings 269
 - 17.6.3 Challenges of Sprint Planning Meetings 270
- 17.7 Sprint Demos 270
 - 17.7.1 Application of Sprint Demos to Distributed Projects 270
 - 17.7.2 Benefits of Sprint Demos 271
 - 17.7.3 Challenges of Sprint Demos 271
- 17.8 Retrospective Meetings 271
 - 17.8.1 Application of Retrospective Meetings to Distributed
Projects 271
 - 17.8.2 Benefits of Retrospective Meetings 272
 - 17.8.3 Challenges of Retrospective Meetings 272
- 17.9 Backlogs 272
 - 17.9.1 Application of Backlogs to Distributed Projects 273
 - 17.9.2 Benefits of Backlogs 273
 - 17.9.3 Challenges of Backlogs 273
- 17.10 Frequent Visits 273
 - 17.10.1 First Visit 274
 - 17.10.2 Further Visits 274
 - 17.10.3 Benefits of Frequent Visits 275
 - 17.10.4 Challenges of Frequent Visits 275
- 17.11 Multiple Communication Modes 276
 - 17.11.1 Benefits of Multiple Communication Modes 276
 - 17.11.2 Challenges of Multiple Communication Modes 277
- 17.12 Conclusions 277
 - References 277
- 18 Feature Teams—Distributed and Dispersed 279**
 - Jutta Eckstein
 - 18.1 Introduction 279
 - 18.2 Context 280
 - 18.3 Historical Structures of Distributed Teams 280
 - 18.3.1 Consequences 281
 - 18.4 Building Agile Teams 281

- 18.4.1 Feature Teams—Co-located or Dispersed 282
- 18.4.2 Creating Proximity for Dispersed Feature Teams 284
- 18.5 Technical Service Team Ensures Conceptual Integrity 285
 - 18.5.1 Starting Team as Role Model 286
- 18.6 Conclusions 286
 - References 287
 - Further Reading 287
- 19 Roles and Responsibilities in Feature Teams 289**
 - Jutta Eckstein
 - 19.1 Introduction 289
 - 19.2 Context 290
 - 19.3 Configuration of a Feature Team 291
 - 19.4 Product Owner 292
 - 19.4.1 Team of Product Owners 292
 - 19.4.2 Lead Product Owner 293
 - 19.4.3 Collaborating with Both: Customers and Feature Team . . . 293
 - 19.5 Coach—Also Known as Scrum-Master 294
 - 19.6 Architect and Architecture 295
 - 19.6.1 Chief Architect 296
 - 19.7 Project Manager 297
 - 19.8 Key Roles Support Their Teams Directly 297
 - 19.9 Conclusions 298
 - References 299
 - Further Reading 299
- 20 Getting Communication Right: The Difference Between Distributed Bliss or Miss 301**
 - Jan-Erik Sandberg and Lars Arne Skaar
 - 20.1 Introduction 301
 - 20.2 Background Overview 302
 - 20.2.1 Background 302
 - 20.3 Starting a Distributed Agile Project 303
 - 20.4 Low-cost and Effective Communication 304
 - 20.5 Empower the Team 306
 - 20.6 Common Architecture Across Locations 307
 - 20.7 On “Proxies” 308
 - 20.8 Conclusions 309
 - References 309
- 21 A Task-Driven Approach on Agile Knowledge Transfer 311**
 - Jörn Koch and Joachim Sauer
 - 21.1 Introduction 311
 - 21.2 Case Overview 312
 - 21.3 Hands-On Approach (Task-Driven Approach) 315

- 21.3.1 Joint Task Planning 316
- 21.3.2 Question-Driven Task Scheduling 316
- 21.3.3 Adequate Task Design 317
- 21.3.4 Scrupulous Task Sign-Off 318
- 21.4 Conclusion 318
- References 319
- 22 Architecture-Centric Development in Globally Distributed Projects . 321**
- Joachim Sauer
- 22.1 Introduction 321
- 22.2 Case Overview 322
- 22.3 Software Architecture and Architecture-Centric Development . . . 323
- 22.3.1 Software Architecture 323
- 22.3.2 Architecture-Centric Development in General 324
- 22.3.3 Architecture-Centric Development in Agile Distributed Settings 324
- 22.4 Distributed Continuous Integration and Collective Ownership . . . 325
- 22.5 Practical Advice for Software Architects 326
- 22.6 Conclusions 328
- References 328
- Part V Epilogue**
- 23 Agility Across Time and Space: Summing up and Planning for the Future 333**
- Darja Šmite, Nils Brede Moe, and Pär J. Ågerfalk
- 23.1 The Beginning of the End 333
- 23.2 Current Themes 334
- 23.3 Practical Advice 334
- 23.4 Areas for Improvement and Future Research 336
- 23.5 The End of The End 337
- Index 339**

List of Contributors

Editorial Board

Darja Šmite is a Senior Researcher at Blekinge Institute of Technology, which has recently been ranked as number 11 among the top institutions in the world in systems and software engineering by the Journal of Systems and Software. She also holds an Associate Professorship from University of Latvia and has been previously engaged in industrial positions at a number of software houses in Latvia before pursuing an academic career. Her major research interests lie in the area of global software development, software process improvement and agile software development. Smite received her Ph.D. from the University of Latvia for her work on addressing the software project risks in globally distributed environment. Contact her at darja.smite@bth.se or darja.smite@lu.lv

Nils Brede Moe is a Research Scientist at SINTEF the largest independent research organisation in Scandinavia. He has 12 years of experience working as a researcher within software development and in consulting software companies around Norway. His research interests include global software development, process improvement, self-management, and agile software development. Moe has a master's of science degree in computer science from the Norwegian University of Science and Technology. Contact him at nilsm@sintef.no

Pär J. Ågerfalk is a Professor at Uppsala University where he holds the Chair in Computer Science in Intersection with Social Sciences. He received his Ph.D. from Linköping University and has held fulltime positions at Örebro University, Lero—The Irish Software Engineering Research Centre, Jönköping International Business School, and University of Limerick, where he is also currently an Adjunct Professor. His work has appeared in a number of leading journals in the software and information systems area, including MIS Quarterly, Information Systems Research, Communications of the ACM, and Information and Software Technology. He is currently the Dean of the Swedish National Research School on Management and IT, a

Senior Associate Editor with the European Journal of Information Systems, a Secretary of IFIP WG 2.13 on Open Source Software, and the founding Chair of the AIS Special Interest Group on Pragmatist Information Systems Research. Contact him at par.agerfalk@im.uu.se.

Contributing Authors

Alberto Avritzer, Siemens Corporate Research, USA

Alberto Avritzer received a Ph.D. in Computer Science from the University of California, Los Angeles. He is currently a Senior Member of the Technical Staff in the Software Engineering Department at Siemens Corporate Research, Princeton, New Jersey. Before moving to Siemens Corporate Research, he spent 13 years at AT&T Bell Laboratories, where he developed tools and techniques for performance testing and analysis. His research interests are in software engineering, particularly software testing, monitoring and rejuvenation of smoothly degrading systems, and metrics to assess software architecture, and he has published over 50 papers in journals and refereed conference proceedings in those areas.

Jan Bosch, Intuit, USA

Jan Bosch is VP, Engineering Process at Intuit Inc. Earlier, he was head of the Software and Application Technologies Laboratory at Nokia Research Center, Finland. Before joining Nokia, he headed the software engineering research group at the University of Groningen, The Netherlands, where he holds a professorship in software engineering. He received a MSc degree from the University of Twente, The Netherlands, and a Ph.D. degree from Lund University, Sweden. His research activities include compositional software engineering, software architecture design, software product families and software variability management. He is the author of a book “Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach” published by Pearson Education (Addison-Wesley and ACM Press), (co-) editor of several books and volumes in, among others, the Springer LNCS series and (co-) author of a significant number of research articles. He is editor for Science of Computer Programming, has been guest editor for journal issues, chaired several conferences as general and program chair, served on many program committees and organized numerous workshops.

Petra Bosch-Sijtsema, Helsinki University of Technology, FINLAND and Stanford University, USA

Petra Bosch-Sijtsema is a senior researcher at Aalto University School of Science and Technology, Laboratory of Work Psychology and Leadership in Finland and currently a visiting scholar at Stanford University, USA, School of Engineering, Project Based Learning Lab, USA. She received her licentiate from Lund University (Sweden) and her Ph.D. from the University of Groningen (The Netherlands) in Management and Organization. She has worked at universities in Sweden, the

Netherlands, Canada, Finland and the US. Her research focuses on innovation, knowledge transfer, management and coordination in global distributed organizations and teams.

Francois Bronsard, Siemens Corporate Research, USA

Francois Bronsard is a Software Engineering Consultant in the Software Development Technologies Group at Siemens Corporate Research in Princeton, NJ. He has a Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign and over 15 years of industrial experience. He has been active in the areas of static analysis, quality assurance, global software development and agile processes for many years and he is a certified SCRUMMaster.

Erran Carmel, American University, USA

Erran Carmel is a Professor at the American University in Washington D.C. His area of expertise is globalization of technology. He studies global software teams, offshoring of information technology, and emergence of software industries around the world. His 1999 book “Global Software Teams” was the first on this topic and is considered a landmark in the field helping many organizations take their first steps into distributed tech work. His second book “Offshoring Information Technology” came out in 2005 and has been especially successful in outsourcing and offshoring classes. He has written over 80 articles, reports, and manuscripts. He consults and speaks to industry and professional groups. He is a tenured full Professor at the Information Technology department, Kogod School of Business at American University. In the 1990s he co-founded and led the program in Management of Global Information Technology. In 2005–2008 he was department Chair. In 2009 he was awarded the International Business Professorship. He has been a Visiting Professor at Haifa University (Israel) and University College Dublin (Ireland). In 2008–2009 he was the Orkand Endowed Chaired Professor at the University of Maryland University College. He received his Ph.D., in Management Information Systems from the University of Arizona; his MBA from the University of California at Los Angeles (UCLA), and his B.A. from the University of California at Berkeley.

Suranjan Chakraborty, Towson University, USA

Suranjan Chakraborty is an assistant professor in the Department of Computer and Information Sciences at Towson University. He also has prior industry experience, having worked for eight years in Wipro Technologies. He completed his Ph.D. in Information Systems from Washington State University. His research interests include requirements engineering, behavioral processes in information systems development, distributed information systems development, and use of qualitative methods in IS research. His research has been published (or accepted for publication) in *Journal of Association of Information Systems*, *European Journal of Information Systems*, *Decision Support Systems*, and *Group Decision and Negotiation*. His work has also been presented or appeared in the proceedings of America’s Conference on Information Systems, Hawaii International Conference on System Sciences, European Conference on Information Systems, and the annual SIG-ED conference.

Kieran Conboy, National University of Ireland Galway, IRELAND

Kieran Conboy is a lecturer in information systems at NUI Galway. His research focuses on agile systems development. Kieran is currently involved in numerous projects in this area, and has worked with many companies on their agile initiatives including Intel, Microsoft, Accenture, HP, and Fidelity Investments. Some of his research has been published in various leading journals and conferences such as ISR, EJIS, TOSEM, IFIP 8.6 and the XP200n conference series. Prior to joining NUI Galway, Kieran was a management consultant with Accenture, where he worked on a variety of projects across Europe and the US.

Jutta Eckstein, IT communication, GERMANY

Jutta Eckstein, a partner of IT communication, is an independent consultant and trainer from Braunschweig, Germany. Her know-how in agile processes is based on over ten years experience in developing object-oriented applications. She has helped many teams and organizations all over the world to make the transition to an agile approach. She has a unique experience in applying agile processes within medium-sized to large distributed mission-critical projects with up to 300 project members. This is also the topic of her books ‘Agile Software Development in the Large’ and ‘Agile Software Development with Distributed Teams’.

Besides engineering software she has been designing and teaching technology courses in industry. Having completed a course of teacher training and led many ‘train the trainer’ programs in industry, she focuses also on techniques which help teach technology and is a main lead in the pedagogical patterns project. She has presented work in her main areas at ACCU (UK), JA00 (Denmark), OOPSLA (USA), XP (Europe) and Agile (USA).

Ann-Dorte Fladkjær Niels, Jyske Bank, DENMARK

Ann-Dorte Fladkjær Nielsen is Project Manager at the Jyske Bank Group. She holds an MSc in Business Systems and Management Engineering, Aalborg University. She has more than 14 years business experience as Project Manager, Project Mentor and Facilitator. She is Certified Project Manager (IPMA level C).

Federico Gobbo, University of Insubria, ITALY

Federico Gobbo owns a Ph.D. in Computer Science since 2009 obtained at the University of Insubria Varese-Como (Italy), where he actually works with a post-doc grant. Since 2005, he has participated to diverse Italian research national projects and he is member of the PASCAL European network of excellence. Before that, he has worked as a web specialist in start-up companies settled in Milan, Italy.

Imed Hammouda, Tampere University of Technology, FINLAND

Imed Hammouda received his Ph.D. in Software Engineering from Tampere University of Technology (TUT)—Finland in 2005. He is currently an adjunct professor at TUT where he is heading the international masters programme at the Department of Software Systems. Dr. Hammouda’s research areas include software architectures, variability management, social software engineering, and open source software development.

Jörn Koch, C1 WPS GmbH, GERMANY

Jörn Koch works as senior software architect at C1 WPS GmbH since 2001. Starting as a developer in 1994 he later got his diploma in computer science and until now gained many years of experience in leading and coaching of agile projects, doing business analysis, and designing and analyzing object-oriented software architectures. From 2005 to the end of 2008 Jörn was a member of the distributed team of the case study's project.

Mikko Korkala, VTT Technical Research Centre of Finland, FINLAND

Mikko Korkala has been involved in agile development since 2002 and is currently working on his Ph.D. thesis on customer communication in distributed agile development. He has been working at VTT Technical Research Centre of Finland since 2007 as a research scientist and has previously worked at the Department of Information Processing Science, University of Oulu, Finland from which he also received his M.Sc. He has also worked as a software engineer. In addition to research, he has provided several agile trainings and workshops and has held invited agile talks both in Finland and abroad. He has also worked as an onsite agile consultant for management in a large software company and has helped to outline agile processes for software companies.

Clifton Kussmaul, Muhlenberg College and Elegance Technologies, Inc., USA

Clifton Kussmaul is Associate Professor of Computer Science at Muhlenberg College, in Allentown, PA. He is also Chief Technology Officer for Elegance Technologies, Inc., which develops software products and provides software development consulting and services. During 2009–2010 he was a visiting Fulbright-Nehru Scholar at the University of Kerala, in southern India. His professional interests include software engineering, free and open source software, scientific computation, and auditory perception.

Michael Lane, University of Limerick, IRELAND

Michael Lane is a lecturer in the department of computer science and information systems at the University of Limerick, Ireland. Michael's research interests revolve around the area of distributed software development. His Ph.D. research is investigating project management in distributed teams leveraging agile software development practices. Prior to joining the University of Limerick in 2005, Michael had spent over 20 years in software development working in various domains. This experience incorporated a number of roles ranging from design and programming of bespoke services in the direct marketing sector to research and development manager of ERP products in the manufacturing sector. His teaching experience includes the delivery of a wide range of subjects to both undergraduate and postgraduate students. Additional educational activities have included the provision of various management training courses.

Casper Lassenius, Helsinki University of Technology, FINLAND

Prof. Casper Lassenius is a professor (pro tem) at the Software Business and Engineering Institute at Aalto University, where he heads the software process research

group. His research interests include software processes, software measurement, quality assurance, and software portfolio and product management.

Likoebe M. Maruping, University of Arkansas, USA

Likoebe M. Maruping is an assistant professor of Information Systems in the Sam M. Walton College of Business at the University of Arkansas. Likoebe's research is primarily focused on the activities through which software development teams improve software project outcomes. His current work in this area focuses on understanding how teams cope with uncertainty in software development projects. He also enjoys conducting research on virtual teams and the implementation of new technologies in organizations. His research has been published or is forthcoming in premier information systems, organizational behavior, and psychology journals including *MIS Quarterly*, *Information Systems Research*, *Organization Science*, *Journal of Applied Psychology*, and *Organizational Behavior and Human Decision Processes*.

Gilberto Mato, Siemens Corporate Research, USA

Gilberto Matos is a Software Engineering Consultant in the Software Development Technologies Group at Siemens Corporate Research in Princeton, NJ. He has a Ph.D. in Computer Science from the University of Maryland at College Park and over 15 years of industrial experience developing end-user applications and software development tools. He has been actively involved in agile and distributed development projects since 2003 and is a certified SCRUMMaster.

Frank Maurer, University of Calgary, CANADA

Dr. Frank Maurer is a Full Professor at the University of Calgary and the head of the Agile Software Engineering (ASE) group at the University of Calgary. His research interests are agile software methodologies, engineering digital table applications, executable acceptance test driven development, integrating agile methods and interaction design, framework and API usability, tools for agile teams, specifically for globally distributed software development and experience and knowledge management. More information about his research can be found at <http://ase.cpsc.ucalgary.ca/>. Currently, the group focuses on empirical investigations of agile techniques, agile product lines, agile interaction design, software design guidelines and application engineering for digital surfaces. He is a member of the Agile Alliance, a Certified Scrum Master, a founding member of the Canadian Agile Network (CAN)—Le Réseau Agile Canadien (RAC), part of the organizers of the Calgary Agile Methods Users Group and Associate Editor of IEEE Software responsible for the Process and Practices area.

Tuomas Mikkola, Solita Oy, FINLAND

Tuomas Mikkola is an Account Manager in Solita working as the supervisor of several software implementation projects and services under maintenance. Previously he has worked as Team Manager and Project Manager in Solita. He received MSc in Software Engineering from Tampere University of Technology in 1999.

Robert Morgan, Red Duck Solutions, CANADA

Robert Morgan received a M.Sc. from the University of Calgary's Department of Computer Science and is a former member of the Agile Software Engineering group. In addition to developing distributed agile planning tools, he has had experience working in the financial and oil and gas sectors as a business analyst, developer and agile champion. He is the founder and CEO of Red Duck Solutions, a Calgary based agile software development and consulting firm. You can visit the web site at: www.redducksolutions.com.

Charles L. Munson, Washington State University, USA

Charles L. Munson is an associate professor in the Department of Management and Operations at Washington State University. His Ph.D. and M.S.B.A. in Operations Management, as well as his B.S.B.A. summa cum laude in finance, are from the John M. Olin School of Business at Washington University in St. Louis. For 2 years he was Associate Dean for Graduate Programs in Business at Washington State University. He also worked for 3 years as a financial analyst for Contel. His research interests include supply chain management, quantity discounts, international operations management, purchasing, and inventory control. Munson has published in journals such as IIE Transactions, Production and Operations Management, Naval Research Logistics, Decision Sciences, European Journal of Operational Research, Journal of the Operational Research Society, Interfaces, Business Horizons, and International Journal of Procurement Management. He currently serves as a senior editor of Production and Operations Management and on the editorial board of the International Journal of Procurement Management.

Karri Mustonen, Solita Oy, FINLAND

Karri Mustonen is a Subcontracting Manager at Solita Oy. He is responsible for building supplier network, supplier development and supplier performance management. He received MSc in Computer Science from Tampere University of Technology, in Finland.

Josy Oliveira, University of Calgary, CANADA

Josyleuda Oliveira is a Ph.D. student in Computer Science at the University of Calgary. Her research interest is agile software methodologies, specifically in globally distributed software development. She received a M.Sc. in Software Engineering from University of Fortaleza-Brazil in 2006. She has had 11 years of experience in industry, in Brazil. She worked as a Project Manager, Software Quality Assurance Analyst and Business Analyst.

Maria Paasivaara, Helsinki University of Technology, FINLAND

Dr. Maria Paasivaara works as a researcher and project manager at the Software Business Engineering Institute at Aalto University. Her main research interest is global software engineering, with a particular focus on collaboration and communication problems and practices.

Minna Pikkarainen, VTT Technical Research Centre of Finland, FINLAND

Minna Pikkarainen has graduated from University of Oulu and has a Ph.D. about the topic of improving software development mediated with CMMI and agile practices. Minna has been working as researcher, project manager and senior research scientist in VTT Technical Research Centre of Finland more than 13 years. During that time she has worked in 18 industrial driven research projects doing close industrial collaboration with more than 15 organizations in Finland, Ireland and Belgium. Minna's research has been published in 25+ journal and conference papers in the forums like ICSE, ICIS and Empirical Software Engineering Journal. So far Minna has provided trainings, workshops and invited talks for 10+ different industries related to agile methods and participated in several conference program committees. Minna has been member of Lero, The Irish Software Engineering Research Centre since 2006. For the past 4 years, her work and publications have been focused on research in the area of agile development.

Rafael Prikladnicki, PUCRS, BRAZIL

Dr. Rafael Prikladnicki is Assistant Professor of Computer Science School at PUCRS. He has been active in the global software engineering (GSE) community for the last nine years and in the agile software development community for the last five years. For the last two years he has been interested in how GSE and agile methodologies impact organizational decisions on software development, including business and technical decisions. He has been acting as coach and instructor in agile software development (focusing on Scrum, XP and Lean). He was member of the organizing committee of the 2009 Latin-American Conference on Agile Development Methodologies, and he is involved with the ICGSE series organizing committee since the first edition in 2006. He is also the general chair of the Brazilian Conference on Agile Software Development, to be organized in 2010. More information online at <http://www.inf.pucrs.br/~rafael>

Jakub Rudzki, Solita Oy, FINLAND

Jakub Rudzki is a Project Manager at Solita Oy. He has worked primarily with distributed Scrum teams. He was also involved in subcontracting initiatives from the beginning at Solita. Jakub Rudzki is a Ph.D. student at Tampere University of Technology, in Finland. His research interests focus on software quality assurance in medium and small software companies. He received MSc in Computer Science from Kielce University of Technology, in Poland.

Cristiano Sadun, Tieto Norway AS, NORWAY

Cristiano Sadun, born 1970 in Milan, Italy, has a degree in Computer Science from the University of Milan, Italy and has been working actively in the software engineering industry since 1990, both as individual advisor and within commercial companies. He has extensive experience with software development, architecture and methodologies, together with business management and sales, primarily in IT consultancy and services areas. He is particularly focused on organizational efficiency, quality of delivery and the continuous improvement of his organization; he

loves to bring together engineering and business perspectives to create value for his company and its employees and customers. When not busy doing all of the above, he can be found playing the guitar at excessive volumes, checking out good restaurants and mostly trying to see the world through the wise eyes of his 10-years old son.

Jan-Erik Sandberg, Det Norske Veritas, NORWAY

Jan-Erik Sandberg has been working as an agile Coach more or less for 10 years now. He currently works as an agile Coach for “Det Norske Veritas”, a world wide organization with more than 200 locations. In 2001 he founded the Norwegian Forum For agile Development. He is an active speaker at many different conferences and seminars, like Microsoft TechEd, Agile200x and XP200X conferences. He is the sponsor chair for XP2010 and has been awarded the “Microsoft Most Valuable Professional” award five years in a row. Jan-Erik is a believer of high quality craftsmanship, pride and enjoyment of work even in large projects.

Saonee Sarker, Washington State University, USA

Saonee Sarker is currently an associate professor in the Department of Information Systems at Washington State University. Professor Sarker received her Ph.D. in Management Information Systems from Washington State University, and an M.B.A. from the University of Cincinnati prior to that. Her research focuses on globally distributed software development teams and other types of computer-mediated groups, technology adoption by groups, technology-mediated learning, and information technology capability of global organizations, and has appeared (or scheduled to appear) in outlets such as Information Systems Research, Journal of the Association of Information Systems, Journal of Management Information Systems, Decision Support Systems, and Journal of Computer-Mediated Communication.

Suprateek Sarker, Copenhagen Business School, DENMARK

Suprateek Sarker is a professor and Microsoft chair of Information Systems at the Copenhagen Business School, Denmark. Until recently, he was Associate Professor and Parachini Faculty Fellow at Washington State University, U.S.A. Much of his research has involved the use of qualitative research approaches, including positivist or interpretive case studies, grounded theory methodology, hermeneutics, and virtual ethnography to study phenomena such as IT-enabled organizational change, ERP implementation, offshoring, and virtual and mobile collaboration. He is currently serving on the editorial boards of journals such as MIS Quarterly, Journal of the AIS, IEEE Transactions of Engineering Management, IT and People, IT for Development, and JITCAR.

Joachim Sauer, C1 WPS GmbH, GERMANY

Joachim Sauer works as software architect at C1 WPS GmbH in the roles of IT consultant, project leader and architect for agile development projects. He holds a diploma in computer science and regularly addresses topics of software engineering and architecture in teaching and research at the University of Hamburg and the HAW Hamburg.

Lars Arne Skår, Miles, NORWAY

Lars Arne Skår—Lars is the CTO of Miles—a Norwegian IT-consulting company focusing on system integration and applying agile practices with established IT departments. Previously he has worked as CTO of a Nordic software development company and as CTO in a Norwegian portal consulting company after some time in an international consulting company. He has been active in the Norwegian agile community which meet regularly at xp.meetup.com in Oslo. As a developer/architect for about 20 years, he is concerned with effective architecture supported by healthy processes. Agile practices are important in this regard, as this has led us back to being conscious about what we really should deliver and engage actively together with the stakeholders both to figure that out and work diligently towards that goal. He has run workshops on former XP conferences (XP2009, XP2008, XP2007, XP2006 and XP2005) as well as on Agile2008.

Jayakanth Srinivasan, Malardalen University, SWEDEN and Massachusetts Institute of Technology, USA

Jayakanth “JK” Srinivasan is a researcher with the Lean Advancement Initiative at MIT, where he focuses on applying and extending lean enterprise thinking to knowledge-intensive industries. His forthcoming book, *Lean Enterprise Thinking: Driving Enterprise Transformation* (co-authored with Debbie Nightingale), presents both the seven underlying principles of lean enterprise thinking as well as field-tested frameworks and tools that organizations can adopt to drive their transformation efforts. His current research focuses on the twin tracks of the sources of enterprise agility in software organizations and the architecture of innovative organizations. Prior to joining MIT, Dr. Srinivasan worked in the public sector on avionics systems and in the private sector writing networking software. His academic training includes a bachelor’s degree in computer engineering, masters degrees in avionics and aeronautics and astronautics respectively, and a doctoral degree in computer science.

Per Svejvig, Aarhus University, DENMARK

Per Svejvig is a Ph.D. student at the Aarhus School of Business, Aarhus University. His research interests are in the area of implementation and use of enterprise systems, managing IT-enabled change, interplay between technology and organizations, and IT project management. He holds a BSc in Engineering, Engineering College of Aarhus and MSc in IT, Aarhus University. He has more than 25 years of business experience as manager, project manager and consultant. He is Certified Senior Project Manager (IPMA level B).

Tarja Systä, Tampere University of Technology, FINLAND

Tarja Systä is a professor at Tampere University of Technology, Department of Software Systems. Her main field of research is software engineering, including e.g. topics related to software development, maintenance and analysis, and software architectures.

Steve Tendon, Agiliter Consultancy Ltd., CYPRUS

Steve Tendon is a Senior Consultant with Agiliter Consultancy, Ltd, Limsassol, Cyprus (<http://agiliter.com>). He has matured more than twenty-five years of professional experience, mainly in the field of software engineering; he is a member of the ACM and of the IEEE. He is currently pursuing a MSc in Software Project Management at the University of Aberdeen. His current interests are in employing innovative or emergent methods and concepts from the fields of services management and systems engineering to help businesses improve their internal and external processes, particularly when bridging software development processes to other business processes and functional areas. You can reach him through email: steve.tendon@gmail.com.

Xiaofeng Wang, Lero, The Irish Software Engineering Research Centre, IRELAND

Xiaofeng Wang is a research fellow in Lero, the Irish Software Engineering Research Centre. Her research areas include software development process, methods, agile software development, and complex adaptive systems theory. Her doctoral study investigated the application of complex adaptive systems theory in the research of agile software development. She has also worked in a research institute in Italy for several years in the area of enterprise knowledge systems. She has published several papers in major Information Systems journals and conferences.

Xin Wang, Ivnet Inc., CANADA

Xin Wang is a software developer for the telephony products at Ivnet Inc. He has written and presented on topics such as using digital tabletops to support agile project planning, the design and implementation experiences on tabletop applications and migrating user interface from desktops to digital tabletops. He received his Msc. in computer science from the University of Calgary in 2009.

Daniel Wildt, FACENSA, BRAZIL

Daniel Wildt is Professor of System Information School at FACENSA. He has been active in the Agile Methodologies community since 2004, leading the Rio Grande do Sul Agile User's Group (Brazil) and acting as coach and trainer in agile methodologies adoption, focused on Lean, eXtreme Programming and Scrum. He was member of the organizing committee of the 2009 Latin-American Conference on Agile Development Methodologies, and is member of the organizing committee of the Agile Brazil 2010, a Brazilian Conference on Agile Software Development. More information online at <http://danielwildt.com>.

Scientific Reviewers

Gabriela Avram

Lero, The Irish Software Engineering Research Centre, IRELAND

Gerry Coleman

Dundalk Institute of Technology, IRELAND

Torgeir Dingsøy
SINTEF, NORWAY

Tor Erlend Fægeri, SINTEF, NORWAY

Alberto Espinosa
American University, USA

Helena Holmström
IT University, SWEDEN

Daniel Luebke
InnoQ, SWITZERLAND

Bala Ramesh
University College London, UNITED KINGDOM

Jonas Sjöström
Uppsala University, SWEDEN

Richard Vidgen
University of Bath, UNITED KINGDOM

Part I
Motivation

Chapter 1

Fundamentals of Agile Distributed Software Development

Darja Šmite, Nils Brede Moe, and Pär J. Ågerfalk

Abstract This chapter provides an introduction to the area of agile distributed software development. It proceeds as follows. We start by introducing and motivating (globally) distributed software development, and follow on with agile software development. With this foundation we discuss the concept of agile distributed development, its motivation and some of the pertinent issues involved.

1.1 Introduction

1.1.1 Distributed Software Development

In the current era of globalization, cross-national and cross-organizational collaboration has become a natural evolution in the operation of the global marketplace. Tight budgets, limited resources and time constraints have motivated many companies to explore global sourcing. This mode of working promises organizations the benefits of reaching mobility in resources, obtaining extra knowledge through recruiting the most talented people around the world, reducing time-to-market, increasing operational efficiency, improving quality, expanding through acquisitions, reaching proximity to market, and many more. As a result, a growing number of software companies have started to implement global supply chains.

D. Šmite (✉)
Blekinge Institute of Technology, Ronneby, Sweden
e-mail: darja.smite@bth.se

N.B. Moe
SINTEF ICT, Trondheim, Norway
e-mail: nilsm@sintef.no

P.J. Ågerfalk
Uppsala University, Uppsala, Sweden
e-mail: par.agerfalk@im.uu.se

While companies are taking the assumed benefits almost for granted, industrial experience shows that these are not as easy to achieve as the literature may lead one to believe [1]. In contrast to other engineering disciplines, developing software is recognized as a significantly complex task that heavily relies on human interaction. Accordingly, distributed software projects with geographically, temporally and socio-culturally dispersed teams unavoidably experience unique pressures and challenges. There are major problems related to communication, coordination and collaboration caused by geographical, temporal and socio-cultural distance.

1.1.2 Agile Software Development

Setting up an agile team is usually motivated by benefits such as increased productivity, innovation, and employee satisfaction. However the agile approach is also motivated by the increasing complexity of software development. As information technology's role in the modern economy has grown in importance, software developers have found themselves confronted with the challenges of exceptional complexity. A software team needs to interact with and consider the viewpoints of a wide variety of stakeholders, many of whom have conflicting views on the software features and functionality. The team must then balance disparate needs of diverse stakeholders, a task far more challenging than merely fulfilling the functional requirements of a system. Thus, it comes as no surprise that many software developers more than welcome agile software development, which embraces these emerging realities at its core.

The agile approach is built around empowered and self-organizing teams that coordinate their work themselves. In such teams there is also a strong focus on collaboration and communication. Collaboration and coordination depend on communication, which is central to successful software development. These activities are supported through various agile practices including pairing, customer collaboration, stand-ups, reviews, retrospectives and the planning game.

1.2 Merging Agility with Distribution

Addressing the problems related to the complexity of software development and the strong focus on collaboration, coordination and communication, are primary reasons for the growing interest in exploring the applicability of agile approaches in distributed software development. Despite the popularity of the topic, the practice of agile development has been well ahead of research in the field [2]. Thus, there is still no consensus or deep, theoretically grounded, understanding of the applicability of agile methods to different types of software projects and the flexibility in application of agile methods necessary to realise the benefits promised.

Table 1.1 Characteristics of agile versus traditional distributed software development

Characteristics	Agile Development	Distributed Development
Communication	Informal	Formal
	Face-to-face	Computer-mediated
	Synchronous	Often asynchronous
	Many-to-many	Tunneled
Coordination	Change-driven	Plan-driven
	Mutual adjustment, self-management	Standardization
Control	Lightweight	Command-and-control
	Cross-functional team	Clear separation of roles

1.2.1 Potential Issues

While the motivation for implementing agility in distributed software development is clear, the process of marrying agility with distribution is not straightforward. Looking at the principles of agile development and the environment of distributed projects, one can easily characterize the two as opposite extremes on a continuum. The fundamental differences between agile and distributed development can be illustrated by the following examples (see Table 1.1).

Methodological standardization has often been argued to be the most effective way to manage global software teams [3]. Consequently, global managers tend to rely on plan-driven methods, in which the life cycle model specifies the tasks to be performed and the desired outcomes of each project phase. These projects are often characterized by defined task division, strict role separation, and preferably complete documentation. Managers are required to perform proper upfront planning and check adherence to the processes through supervision.

However, because of geographical, temporal and socio-cultural distance, standardization and command-and-control oriented management often fail. Distributed development is associated with computer-mediation, asynchronous communication, and lack of transparency for remote activities. Thus, the applicability of these coordination mechanisms is typically insufficient. This motivates the application of agile methods based on mutual adjustment and teamwork for distributed project coordination.

These differences between agile and distributed foundations suggest that the application of agile methods in distributed environments is doomed to fail. In fact, many believe that being agile and distributed is unrealistic. For example, Kontio et al. [4] claim that agile practices are hard to implement in distributed teams especially when the team size is large. On a similar note, Taylor, Greer et al. [5] claim that distributed agile software development suffers substantial difficulties because of its complex development environment and lack of empirical evidence describing the actual development experiences.

Due to geographical separation of stakeholders and teams in distributed projects, many of the fundamental concepts promoted by agile approaches are indeed difficult

to apply. The implementation of such practices as pair-programming, shared code ownership and onsite customer, puts demands on the distributed projects and leads to tailoring the practices and compensating the lack of co-location and face-to-face interaction through innovative information technology and communication tools. However, it has been argued that there are issues that cannot be solved through tools. For example, it is widely believed that trust needs touch. Accordingly, the application of agile methods for distributed projects, and thus the extent to which the benefits of agility can be achieved, may be hampered.

1.2.2 All or Nothing versus Á la carte

While agile methods promote flexibility, some agilists advocate an all-or-nothing attitude to the selection and application of the methods and practices. This is because only the synergetic combination is recognized to guarantee the maximum benefits. Hence, the tailoring attitude to agile methods is often not very well received. The question of the viability of agile approaches in distributed projects then certainly springs to mind. If distributed projects are forced to select and tailor agile methods and practices, therefore following á la carte approach, what are the benefits that one can expect? With this aim a series of studies have been conducted and experiences clearly show that agility across time and space not only exists but also gains success. As long as developers understand the rationale behind certain practices suggested by a method, tailoring or even replacing parts should not be a problem. The issue at stake is rather to make sure that this rationale is communicated to developers for them to act upon in an informed way.

1.3 Current Practice

Previous work and published experiences (e.g. by some of the authors and editors of this book) show successful implementation of agile values and principles in different distributed projects. This motivates the assessment of the viability of agile practices for distributed software development teams. Working on this book we have sought to understand the major areas of interest covered by current research. With this aim, we conducted a literature study and identified 41 relevant research papers from the following conferences: XP, ICGSE, Agile, Euromicro, HICSS, COMPSAC, ASPEC and EuroSPI. Research topics covered by these articles included:

- Benefits of introducing agility in distributed projects;
- Adopting agility for distributed projects;
- Communication in agile distributed projects;
- Planning and coordination in agile distributed projects;
- Customer relationship in agile distributed projects;
- Tool support for implementing agility in distributed projects;

- Recommendations for implementing agility in distributed projects;
- XP in distributed projects;
- Scrum in distributed projects;
- User stories in distributed projects.

These topics illustrate three trends. First, researchers are exploring the benefits of agile methods in distributed environment. Second, considerable effort is put into exploring the practices that unavoidably require tailoring through, for example, tool support. Finally, a large number of articles are dedicated to investigating the best practices for implementing agility, including attempts of validation of selected agile methods (or parts thereof) in distributed environments.

1.4 Conclusions

Arguably, the combination of agile and distributed development is of immense interest to industry. Agile development practice has always been ahead of research, with academics struggling to understand what is going on and why it apparently works so well. The same is true about agile distributed development, where practitioners started to experiment and quickly adjust their strategies. As a result, a number of agile methods have been tried out in distributed projects, and there is certainly a lot to be learnt from that experience.

Despite the popularity of the topic, we still do not understand fully the limitations and viability of agile methods in seemingly incompatible environments of distributed software projects. Agile methods work well in the settings they were designed for (i.e. small co-located teams). How they will play out in large, globally distributed projects, is still an open question. However, the remainder of this book is devoted to explore this question and provide actionable advice for anyone embarking on such a quest, or is just interested in the area for any other reason (including being a researcher, perhaps).

References

1. Ó Conchúir, E., Ågerfalk, P. J., Fitzgerald, B., & Holmström Olsson, H. (2009). Global software development: Where are the Benefits? *Communications of the ACM*, 52(8), 127–131.
2. Ågerfalk, P. J., & Fitzgerald, B. (2006). Flexible and distributed software processes: Old petunias in new bowls? *Communications of the ACM*, 49(10), 26–34.
3. Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Englewood Cliffs: Prentice-Hall.
4. Kontio, J., Hoglund, M., Ryden, J., & Abrahamsson, P. (2004). Managing commitments and risks: challenges in distributed agile development. In *Proceedings of the international conference on software engineering* (pp. 732–733).
5. Taylor, P. S., Greer, D., Sage, P., Coleman, G., McDaid, K., & Keenan, F. (2006). Do agile GSD experience reports help the practitioner? In *Proceedings of the 2006 international workshop on global software development of ACM* (pp. 87–93).

Part II

Transition

Chapter 2

Implementing Extreme Programming in Distributed Software Project Teams: Strategies and Challenges

Likoebe M. Maruping

Abstract Agile software development methods and distributed forms of organizing teamwork are two team process innovations that are gaining prominence in today's demanding software development environment. Individually, each of these innovations has yielded gains in the practice of software development. Agile methods have enabled software project teams to meet the challenges of an ever turbulent business environment through enhanced flexibility and responsiveness to emergent customer needs. Distributed software project teams have enabled organizations to access highly specialized expertise across geographic locations. Although much progress has been made in understanding how to more effectively manage agile development teams and how to manage distributed software development teams, managers have little guidance on how to leverage these two potent innovations in combination. In this chapter, I outline some of the strategies and challenges associated with implementing agile methods in distributed software project teams. These are discussed in the context of a study of a large-scale software project in the United States that lasted four months.

2.1 Introduction

It is not a surprise that constructing software applications is an inherently complex task that requires significant coordination and project management resources. We repeatedly read it in the academic and practitioner literature in information systems and software engineering. The construction of software itself requires significant expertise to be brought to bear. More often than not, these software construction efforts must be orchestrated among developers whose work tasks are highly interdependent. In addition to being well-versed in managing the technical domain of software development, developers need to have some understanding of the business domain, for which the software is being constructed. As if these challenges were not

L.M. Maruping (✉)

Sam M. Walton College of Business, University of Arkansas, Fayetteville, AR 72701, USA
e-mail: lmарuping@walton.uark.edu

enough, the past two decades have seen the rapid evolution of the business environment for software products. In particular, the cycle times for delivering functional software have become increasingly aggressive. Further, the very nature of what the software product is expected to do—that is, the business requirements—have been changing more frequently. As the introduction to this book has already highlighted, agile methods have emerged as a highly effective approach for meeting the challenges posed by today’s demanding software development environment.

Significant advances in the capabilities of information and communication technologies (ICTs) have enhanced the capacity of organizations to draw on software development expertise regardless of where it is physically located. An obvious advantage provided by such reach is that organizations can more readily draw upon the expertise needed to solve business problems as they arise. Consequently, we are seeing the increased use of distributed forms of organizing software project teams [4]. Not surprisingly, given the promise of agile methods for software project performance, there has been tremendous interest in understanding how organizations can gain the benefits of implementing agile practices in distributed software project teams [2, 3, 8]. However, distributed forms of organizing software project teams present their own challenges in terms of communication, coordination, and control [5, 12]. Distributed software project teams can also differ substantially from each other in terms of their degree of spatial, temporal, and configurational dimensions—each of which affects the ability to implement key processes [9]. Unfortunately, adopting agile methods in distributed software project teams is not a straightforward process. Each agile method comprises different practices. These practices differ in the extent to which they can be adopted in a distributed context. Consequently, there is value in examining strategies for implementing agile methods at the level of the individual practices. In fact, this process of method tailoring and utilizing an à la carte approach to adopting agile method practices is more the norm than the exception in organizations. With these issues in mind, my goal in this chapter is to outline some strategies for implementing agile practices in a distributed software project team setting. The suggestions provided here are based on observations from a four-month study of 56 software project teams following the eXtreme Programming (XP) method.

2.2 Implementing XP Practices: Where Is an Organization to Start?

2.2.1 The Promise of XP

Among the various agile development methods that have emerged in response to the evolving environment for software development, XP has been one of the most widely adopted. Consistent with other agile methods, and as stated in the agile manifesto, XP places an emphasis on individuals and interactions over processes and

tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [7]. The XP method promises a number of benefits to software project teams including the production of higher quality functional software, shorter development cycles, satisfied customers, and less stressed developers. Evidence from academic research has pointed to the success of XP in yielding positive project outcomes—particularly under conditions where project requirements tend to be unstable (e.g., [6, 10, 11]).

While there is an abundance of guidance on how to implement XP in software project teams whose members are co-located, managers find themselves facing uncertainty about how to implement XP in a distributed setting. Views on whether XP can be successfully implemented in distributed software project teams are currently mixed. For instance, Batra [3] suggests that the structure of distributed settings presents challenges that preclude the possibility of successfully implementing XP in global software project teams. In contrast, Holmström et al. [8] suggest that agile methods such as XP can actually reduce the challenges posed by distributed settings. In order to understand the issues surrounding the implementation of XP in distributed project teams, it is important to focus the discussion on the practices that make up the XP method. This is critical because XP practices differ in their functional purpose and the conditions required for their successful implementation. For example, some XP practices are more planning-oriented in nature whereas others are more action-oriented. Furthermore, a practice-oriented approach resonates quite well with the prevailing à la carte approach utilized by most organizations implementing XP [6, 10].

2.2.2 Understanding How Your Software Project Team Is Structured and Why It Matters

It is important that managers be aware of precisely how their software project teams are distributed. Team distribution is not a unitary concept. Rather it encompasses several dimensions—each of which carries different implications for software project teams' ability to effectively execute certain processes. Team member distribution can be characterized in terms of spatial, temporal, and configurational dimensions [9].

Spatial dispersion represents the average distance between the physical locations in which team members are situated. The spatial dispersion of teams can range from having all team members working in the same building on the same floor to a situation where team members are spread across different countries. When quantified in measurement terms, this can range from distances of a mere few meters to distances of thousands of kilometers. Viewed from this perspective, it is clear to see that there are various ways in which the distance can be traversed to facilitate communication among team members. In teams with low spatial dispersion, this can be as simple as walking down the corridor to another teammate's office. In teams with slightly higher spatial dispersion it could involve driving a few hundred kilometers to the next city. In more extreme cases it can involve taking an overnight

flight to a different country. Of course, advanced ICTs (e.g., VoIP, Skype, Instant Messenger, videoconferencing) now exist to facilitate inexpensive and high quality communication regardless of distance.

Temporal dispersion reflects the extent to which there is overlap in normal work hours for team members. This is conceptually different from spatial dispersion. Two teams can have the same level of spatial dispersion but have different levels of temporal dispersion. For instance, one team might have its members spatially dispersed across Boston and Atlanta. This team would be considered to have a high level of spatial dispersion. However, all team members operate within the same time zone and therefore have completely overlapping work hours—i.e., low temporal dispersion. Another team might have its members spatially dispersed across Boston and Omaha. Arguably the level of spatial dispersion is similar for both teams. However, the two locations operate in different time zones. Therefore, this team has a higher level of temporal dispersion than the first one. In general terms, teams whose membership is distributed in an East-West orientation are more likely to be temporally dispersed (i.e., span different time zones) than teams whose membership is distributed in a North-South orientation (i.e., within the same time zone). Increasing levels of temporal dispersion make it challenging for teams to engage in effective coordination of work.

Finally, *configurational dispersion* is the arrangement of team members across physical locations. These locations can be thought of in terms of the number of distinct buildings or cities where a team's members are located. The configurational dispersion of a team can be understood independently of spatial and temporal dispersion. For instance, two six-member teams can have their members spread across Geneva and St. Gallen. The first team might have three members located in the Geneva office and three members in the St. Gallen office. In contrast, the second team might have one member located in the St. Gallen office and five members in the Geneva office. Clearly both teams have the same level of spatial and temporal dispersion. However, the arrangement of team members across locations differs. Certain configurations of dispersion have been found to be detrimental to team functioning [5]. Table 2.1 below includes a summary of these different forms of dispersion and provides examples of teams exhibiting low and high levels of each.

By understanding the dimensions, along which their software project teams may be distributed, managers are in a better position to understand what challenges and opportunities exist for successfully implementing XP practices and enjoying the benefits those practices provide. In the remainder of this chapter, I briefly introduce the setting, from which my observations are drawn. I then describe the strategies utilized and pitfalls to be avoided in implementing the XP practices within the context of specific forms of team distribution.

2.3 Case Overview

I conducted a four-month study of a large-scale software development project in a large U.S.-based software organization. The organization operated in both the pack-

Table 2.1 A summary of different forms of team dispersion

Form of Dispersion	Definition	Examples
Spatial	The average distance between the physical locations in which team members are situated	<i>Low:</i> A team where members are located within close proximity of each other (e.g., the same building) <i>High:</i> A team where members are located in different countries (e.g., across an office in the U.S. and an office in Japan)
Temporal	The extent to which there is overlap in the normal work hours for team members	<i>Low:</i> A team where members are located in the same time zone (e.g., across an office in Toronto and an office in Miami) <i>High:</i> A team where members are located in different time zones (e.g., across an office in Toronto and an office in Milan)
Configurational	The arrangement of team members across physical locations	<i>Low:</i> A team whose members are equally spread across three sites (e.g., three members at each site: 3-3-3) <i>High:</i> A team whose members are unevenly spread across three sites (e.g., two members at the first site, one member at the second site, and six members at the third site: 2-1-6)

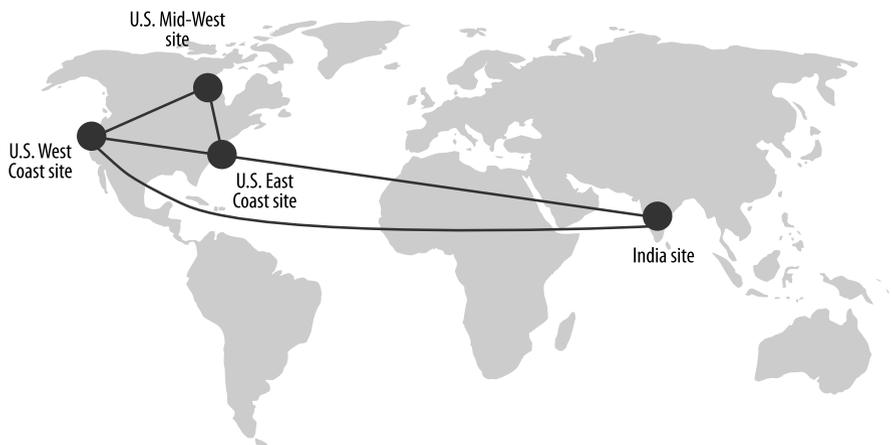


Fig. 2.1 Geographic dispersion of the software project teams

aged software and the custom-built software markets. Given its established operations in different locations across the globe (e.g., India, South Africa, China), the

organization had already gained extensive experience with managing distributed project teams. In contrast, agile methods had only been adopted by the company within the last few years. The project involved 73 different software project teams working independently to create an enterprise-wide software application designed to support business processes in a U.S.-based customer organization. The sizes of the project teams ranged from eight to twelve members. As Fig. 2.1 illustrates, most of the project teams were composed of members who were located within the U.S. However, a proportion of the project teams had members spread across the U.S. and India. Even within the U.S., project teams had members spread across the West Coast, the Mid-West, and the East Coast. The average age of team members was 29.4 and they had an average of 7.2 years of programming experience. The project teams had a variety of ICTs at their disposal, including videoconferencing, chat, email, and telephone. In addition to having its own assigned leader, each project team was also assigned a representative from the customer organization. The customer representative assigned to each team was an expert in the specific business domain for which the software module was being developed.

Each project team was assigned to design and create a specific module of the system. Although the participating organization encouraged the use of XP practices by its project teams, team leaders had the autonomy to decide which specific XP practices their team used. Consequently, teams varied in the extent to which they implemented various XP practices. It is important to note that this is not unlike observations from other studies of XP use in other organizations (e.g., [1, 10, 11, 13]). As noted earlier, the project spanned a four-month period. This afforded me an opportunity to observe various phases of the projects unfold over time. All 73 project teams began work on their assigned modules at the same time. Given the number of teams involved in the study, I primarily employed a survey-based methodology for assessing the level and timing of XP practice use. These assessments were conducted at three different points in time over the course of the project timeline. Out of the 73 project teams involved, 56 participated in all phases of the survey assessment. Consequently, my observations are based on these 56 project teams. In the discussion that follows, I draw on observations of teams in this project to outline recommendations for appropriation of XP practices in distributed software project teams. Table 2.2 provides a summary description of the project and project teams.

2.4 XP in Distributed Software Project Teams: Implementation Strategies and Pitfalls to Avoid

2.4.1 *The Planning Game*

Motivation The planning game is a critical process for understanding the requirements for the software application under construction. This practice is used to outline the scope of the next release of the software, including core features to be included and a rough estimate of the timeline for completion. An important benefit of

Table 2.2 A summary of project features

Project/Team Feature	Description/Statistic
Project duration	Four months
Number of project teams	73 teams
Number of developers	689 developers
Average age of team members	29.4 years
Average project team size	11 members (range: 8–12 members)
Average development experience	7.2 years
Locations involved	India, U.S. West Coast, U.S. Mid-West, U.S. East Coast
Implemented practices	Planning game (approx. 48 teams), collective ownership (approx. 35 teams), coding standards (approx. 40 teams), use of metaphor (approx. 10 teams), simplicity of design (approx. 36 teams), sustainable pacing (approx. 27 teams), pair programming (approx. 40 teams), continuous integration (approx. 36 teams), unit testing (approx. 36 teams), refactoring (approx. 29 teams), customer involvement (approx. 28 teams), small releases (approx. 37 teams)

this practice is that it enables project teams to visualize the business process that will be supported by the features and functions of the next release. This is accomplished by working closely with the customer representative and laying out the story cards in a sequence that reflects business processes of interest.

Implementation Description An important part of implementing the planning game is the use of story boards to outline the sequence of activities that make up the business process being supported. This typically involves a large physical space, on which index cards can be placed and moved around. In a distributed setting such an approach may not make much sense since not all developers are able to physically work with that space. Some of the more successful project teams in the study reported replicating this process electronically by using a central digital repository for the story boards, which all team members had access to. This shared space allowed all team members to view and manipulate the story boards in real-time.

Limitations The degree of spatial and configurational dispersion did not seem to affect the ability of project teams in the study to successfully execute the planning game. In some cases, high levels of temporal dispersion presented a few challenges. One team that had team members spread across India and the East and West coast of the U.S. complained that it was often difficult to reach a shared understanding about the next release because team members could never all meet at the same time and discuss issues pertaining to the story boards. Often, if team members in the U.S. had met and agreed upon some elements on the story board, the team members in India would be left to interpret what was in the repository. It often took a significant amount of back-and-forth discussion through email and late night (or early morning) phone calls to resolve inconsistencies in understanding across sites.

Practical Tips

Tip 1 For highly temporally distributed project teams it would probably be beneficial to hold periodic team meetings to discuss the story boards involved in the next release of the software.

Tip 2 Planning game meetings will be more effective if conducted using synchronous communication such as telephone or videoconference. Chat software may also work but can be prone to miscommunication. With chat software it may also take longer for team members to reach a common understanding of issues.

2.4.2 Collective Ownership

Motivation Collective ownership is an important project team practice because it provides a clear understanding of the roles and responsibilities of all team members with respect to the software. This practice has been found to enhance the quality of software produced by project teams. By giving all team members a shared responsibility for the software, collective ownership encourages quality-enhancing practices such as refactoring.

Implementation Description Many of the teams that implemented the collective ownership practice only needed to meet once during the early phases of the project to discuss roles and responsibilities. For the duration of the project the sense of collective ownership, and the activities it enabled, were reinforced from time to time. Team members at each site understood that if they needed to make changes to any part of the software, they were well within their rights to do so. It is important to note that action-oriented practices such as unit testing, acceptance testing, and continuous integration served as an important safety net for teams using the collective ownership practice. Through these action-oriented practices, project teams were able to identify defects or deviations from customer requirements that may have been introduced by a team member who was making changes to the software code.

Limitations Because collective ownership pertains to roles and responsibilities, none of the three forms of team dispersion posed a challenge to the project teams in the study. Distributed teams generally did not have any problems implementing this practice.

Practical Tips

- Tip 1 If you are considering implementing the collective ownership practice, communicate the roles and responsibilities clearly during the early phases of the project. Consider having a team meeting involving all sites for this discussion.
- Tip 2 Clearly outline the norms surrounding the behavior encouraged by collective ownership. For instance, if any member can modify any part of the code at any time, it would be important that other team members be notified when such a change is being effected. If not, then safe-guards need to be put in place to prevent defects from inadvertently being introduced.

2.4.3 Coding Standards

Motivation The use of coding standards has tremendous benefits for software project efficiency and quality, especially when the teams are composed of distributed members. By having an agreed upon set of standards (e.g., about variable naming conventions, data types, coding structures), team members have a shared understanding for interpreting each other's work. Project teams find it easier to communicate and coordinate when such standards are in place.

Implementation Description Project teams that implemented the coding standards practice used two mechanisms. First, during the team kick-off meeting, the standards were communicated and discussed by all team members. Some teams conducted the meeting using telephone conferencing while others employed video-conferencing technology. This allowed team members to seek clarification and elaborate on various issues effectively. The use of these technologies minimized the potential for miscommunication and misunderstanding. Second, the coding standards that were agreed upon were documented and made available to all team members via a shared digital repository. Members at each site were able to access this repository at any time.

Limitations Spatially and configurationally dispersed project teams did not face any challenges with implementing the coding standards practice. In general, temporally dispersed project teams did not face any challenges either, provided that all team members were involved in the initial meetings to discuss issues pertaining to coding standards.

Practical Tips

Tip 1 Coding standards should be discussed and established early in the software project lifecycle. Managers should make sure that all members of the project team participate and understand the standards that will guide their work.

2.4.4 Use of a Metaphor

Motivation Metaphors provide a useful guide to team members on how the system, as a whole, should operate. The use of metaphors enables team members to develop a mental map of the system. As a result, team members are able to understand how their inputs fit into the big picture.

Implementation Description Only a few project teams in the study implemented the metaphor practice. This is consistent with other observations of XP in the field. The use of metaphor tends to be the least commonly followed practice. One team that implemented the metaphor practice held a team meeting at the start of the project. The team leader solicited the input of team members on what the metaphor should be. Team members engaged in a lively discussion of the metaphor via videoconference. In one case, the team members at the India site were not familiar with one of the metaphors proposed, so members at the U.S. site spent time explaining the metaphor until it was understood. Once all team members had a good understanding of the metaphor, they adopted it and used it as a guide for the project.

Limitations Due to the availability of tele- and videoconferencing technology, spatial and configurational forms of dispersion did not present much of an impediment to software project teams. Temporally dispersed teams faced some minor challenges. These challenges occurred in teams that were composed of members in the U.S. and India and were largely linked to cultural differences in the understanding of specific metaphors. Differences needed to be resolved via synchronous communication.

Practical Tips

Tip 1 Consider the composition of your team when implementing the metaphor practice. Are all team members familiar with the metaphor? Does the metaphor hold the same meaning for all team members? This may be especially important if your team has members in another country that is culturally different from yours.

2.4.5 *Simplicity of Design*

Motivation The emphasis of the simple design practice is on developing software functionality using the simplest coding structure possible to make it work. Simple software design yields multiple benefits for project teams. It reduces the potential for software defects, reduces the amount of effort that team members must expend to understand the code, and it makes it much easier to implement changes to the code when necessary. Simple design is especially critical for distributed software project teams since development efforts need to be coordinated between team members who have a lack of shared context. Communication and coordination are much easier to manage when the design of the software is simple.

Implementation Description Many software project teams found it relatively easy to implement the practice of simple design. During the early phases of the project, teams discussed the methods through which simplicity would be achieved in the design of the code. Much of this was articulated in the coding standards used to guide the team's development effort. Teams were careful to focus their development effort on features and functionality related to the current iteration of the software and not on anticipated (future) iterations. A few teams sought to simplify the design by minimizing the number of classes and methods in the software code.

Limitations As a guiding principle for the structure of the software, spatially, temporally, and configurationally dispersed project teams were unobstructed in their ability to implement the simple design practice. In fact, the implementation of this practice yielded significant benefits in temporally and configurationally dispersed project teams because it reduced the associated coordination challenges.

Practical Tips

Tip 1 Managers should encourage project teams to implement the simple design practice early in the project life cycle. Adherence to, and reinforcement of, this practice will facilitate behaviors such as refactoring that will yield benefits for the duration of the project.

Tip 2 The simple design practice will be especially beneficial in project teams that are highly temporally and/or configurationally dispersed. These types of project teams face considerable coordination challenges. The simple design practice reduces complexity in the structure of the software code, making coordination more manageable.

2.4.6 Sustainable Pacing

Motivation Sustainable pacing provides guidelines about how to manage the timing of code development, testing and deliverables. Developing software to meet a customer deadline is undoubtedly demanding on project team members. The process of developing software is, by nature, a complex and knowledge-intensive undertaking. Coordinating team member inputs across multiple sites makes this process even more demanding. Team members often log a large number of hours, working late to produce functional software within an established project deadline. Such demands inevitably create stress, exhaustion, and burnout among team members. Developing software code in such a state increases the likelihood of defects being introduced or overlooked. The principle of sustainable pacing ensures that team members work on a comfortable schedule that can be maintained for the duration of the project. With sustainable pacing, developers are able to bring more energy to their work on the project because they do not have to work for an unreasonable number of hours.

Implementation Description The implementation of the sustainable pacing practice was largely driven by project parameters such as release schedules, module size, and project team size. For instance, one project team was responsible for designing a module to support the customer's procurement processes. Given the complexity of the procurement process itself, coupled with the importance of the process to the customer's business operations, a larger number of release cycles were required. Consequently, the project team had little control over the pacing of project deliverables. In contrast, another project team was responsible for developing the customer's HR module for processing employee reimbursements. This project required fewer release cycles and the project team had flexibility in determining the timing of the releases. Interestingly, project teams that had members located at U.S. and Indian sites were able to facilitate sustainable pacing more effectively than project teams with members located in the U.S. only. This was partly because the temporally dispersed teams were able to send work from one site to another at the end of the work day, ensuring that progress continued to be made in achieving project goals.

Limitations Project teams were generally quite limited in their ability to implement the sustainable pacing practice. Their ability to implement this practice was contingent on project-related constraints. Within these constraints, temporally dispersed project teams were found to be more effective in implementing this practice.

Practical Tips

Tip 1 Although control over sustainable pacing is largely determined by project parameters and customer deadlines, managers can enhance the ability to manage this practice through the design of the team. Consider composing the team of members who are located across different time

zones. Alternatively managers can compose larger teams to spread the workload. However, the benefits of this approach need to be weighed against the added cost of additional employees devoted to the project.

2.4.7 Pair Programming

Motivation As earlier chapters have already noted, the pair programming practice in XP involves two developers using one computer to write code. Roles in this practice include one developer writing software code while the other developer conducts a real-time code review-identifying errors as they occur-and broadly maps out tests for the code. The roles rotate between the developers involved. Pair programming yields numerous benefits, including the ability to produce more software code of a higher quality.

Implementation Description The ability to implement the pair programming practice was contingent on the design of the project team. Project teams that were spatially or temporally dispersed were able to institute the practice by co-locating developers at each location. Through this structure, pair programming could be executed in much the same way as it would be in a co-located team. Pair programmers could then coordinate their work with pairs at other physical locations (and in other time zones). For example, one project team that was responsible for developing the module to support customer billing was composed of four members at the east coast site in the U.S., and five members at the Indian site. This team used the pair programming practice within geographic sites and then coordinated their work across sites. Consequently, the project team as a whole was able to realize the benefits of pair programming.

Limitations Configurational dispersion posed a major challenge to some project teams. In particular, project teams that had an isolated team member were somewhat constrained in their ability to fully implement pair programming. One such team attempted to manage pair programming through electronic means. Much of the communication was conducted through a chat application and telephone. This approach proved to be highly ineffective as the team experienced numerous delays, frequent misunderstandings about the code, and a significant amount of frustration among team members. After several unsuccessful attempts, the project team opted to change the role of the isolated developer and instead manage all pair programming at sites with co-located team members only.

Practical Tips

- Tip 1 To the extent possible, avoid distributed pair programming. Depending on the design of your project team, consider implementing pair programming within sites rather than across sites. This will enable your project team to reap the benefits of pair programming much more effectively.
- Tip 2 Manage the coordination of project inputs between pairs of programmers across sites. Such an arrangement will reduce the complexity of managing multiple interdependencies among different team members within and across development sites.

2.4.8 Continuous Integration and Unit Testing

Motivation Tests and continuous integration are often tightly intertwined. Continuous integration yields several benefits for project teams. Through this practice, project teams are able to produce and maintain an application with minimal code defects. This is because changes to the software code are not committed to the production code until the newly integrated software passes all functional tests. When defects are discovered, it is easier to trace the changes to which those defects are linked. This iterative approach to development also makes it easier to incorporate changes to the software and/or add new functionality with minimal effort.

Implementation Description Project teams that implemented the continuous integration practice were able to work across geographic and temporal boundaries effectively. Two different approaches were used. In one team, continuous integration was implemented for components that were developed within the U.S. West coast site. Simultaneously, continuous integration was used in developing components at the Indian site. Once their code passed all functional tests, team members at the Indian site then handed off their component to the U.S. site. Integration of both components was then conducted, including associated testing and modification. In an alternative approach, another project team distributed the continuous integration process across sites. While the integration of the code was managed at the U.S. East coast site, the testing and identification of defects in the integrated code was managed at the Indian site. Efforts to eliminate the defects were then managed at the U.S. site and the process would be repeated. Both approaches proved to be successful in curbing the emergence of defective software code.

Limitations Spatial and temporal dispersion were not a barrier to effective implementation of continuous integration. Configurational dispersion was also not a

challenge. However, project teams that were distributed across three sites needed to be more careful about how they coordinated the handoffs and responsibilities in this process. In a few cases there was some confusion about which site was responsible for implementing changes that were identified through testing. Duplication of effort occasionally resulted from this confusion.

Practical Tips

Tip 1 When implementing continuous integration with multiple sites involved, it is important to be very clear about the roles and responsibilities of each site. It is also important to be explicit about where handoffs will occur. It is preferable for handoffs to occur between sites because each site then has a clear understanding of its role and responsibilities.

2.4.9 Refactoring

Motivation The refactoring practice enables project teams to develop efficient software code that is easy to comprehend. The elimination of duplicate code reduces the potential for defects. Efforts to simplify the structure of the code enable project teams to more effectively incorporate changes to the design if and when it becomes necessary to do so. As a result, the costs of making changes to the software at later stages of the project life cycle are significantly reduced. This practice reinforces the principle of simple design.

Implementation Description Project teams that implemented the refactoring practice successfully were able to leverage the fact that the practice itself represents a dyadic relationship between a developer and the code. Project team members at each site had access to the code from a central repository. Therefore, when an opportunity to enhance the code was identified, a developer simply downloaded a copy of the baseline code from the repository and made enhancements to the copy. Once enhancements were completed, the test code was uploaded to the repository. Team members at other sites could then conduct the necessary testing and integration before the enhancements were accepted and committed to the production code. As in the case of collective ownership, testing and continuous integration served as important safety nets to prevent the introduction of defective code.

Limitations Spatial, temporal, and configurational dispersion did not present any challenges to teams that implemented the refactoring practice. Occasionally, highly temporally dispersed project teams needed to expend additional effort to communicate via telephone or videoconference when clarifications were needed across sites. For instance, in one case developers at the Indian site were not sure if a calculation for raw material transportation costs were measured in pounds (lb) or kilograms

(kg). The developers at the U.S. site assumed the measurement was in pounds but never communicated this to the developers at the Indian site. A conference call between developers at the U.S. site and the Indian site was conducted to resolve the ambiguity before enhancements to the software were made.

Practical Tips

Tip 1 Take great care in implementing refactoring across geographically distant sites. Managers should ensure that the implementation of this practice is coupled with systematic checks through testing and continuous integration. This will reduce the need for developers to notify each other when making enhancements to the software code.

2.4.10 Customer Involvement

Motivation Having the dedicated attention of a member of the customer organization has numerous benefits. Project teams are able to get a clearer understanding of project requirements beyond what is included in requirement documentation. Customer representatives on the team have a much better understanding of the business environment in which the software will be used. They can quickly resolve ambiguities about requirements and can give immediate feedback on design issues. Customer representatives also perform a critical role in writing acceptance tests. Such tests ensure that the software actually meets customer needs.

Implementation Description The customer representative was physically located at one of the U.S. sites. Therefore, team members who were co-located with the customer representative were able to share information with team members at other sites. Occasionally, when team meetings were required, the customer representative would be located at one site and members at other sites would communicate via telephone or videoconference. For instance, the project team responsible for the billing module assigned the interface design to developers at the U.S. sites and assigned much of the coding for data processing to the developers at the Indian site. Developers at the Indian site would stay at the office late to participate in the team meeting with the customer and developers at the U.S. site. Through this approach, all team members were able to gain clarity on various issues pertaining to customer requirements as well as receive feedback on results of acceptance tests. Of course, results of acceptance tests could also be documented and sent to project team members without need for a formal team meeting.

Limitations As long as project teams were able to use synchronous communication media, spatial, temporal, and configurational dispersion did not present any

challenges to project teams that sought to leverage their client members' knowledge and familiarity with the organizational context.

Practical Tips

Tip 1 To the extent possible, managers should ensure that at least one project team site is co-located with the customer. This will facilitate better customer-to-project team knowledge sharing through face-to-face interaction. Synchronous communication media can be used to facilitate site-to-site transfer of information.

Tip 2 Managers should also try to arrange periodic team meetings that involve the client member. This will ensure that developers at other sites remain in tune with the customer's needs as the project progresses.

2.4.11 Small Functional Releases

Motivation The practice of deploying small releases of functional software reflects a process for delivering the product to the customer. This approach enables project teams to focus on first delivering the most critical functionality to the customer. Other important components of the software can then be added iteratively in subsequent releases. This approach also provides flexibility for the project team to incorporate feedback from users in the customer organization since suggestions can always be included in the next release. This iterative approach to delivering functional software often yields high customer satisfaction.

Implementation Description Project teams deployed releases of the software at the customer site. Since the project teams had a physical presence in the same geographic location as the customer organization, the deployment of releases was executed easily. Early releases encompassed the critical functionality. Subsequent releases included functionality that was needed but not high priority. Small releases were deployed after extensive unit and acceptance testing. The release cycles for each development team differed according to what made sense for the module assigned. For instance, for modules that required many different features, development teams tended to have shorter release cycles. Generally, the practice of deploying small releases proved to be beneficial for the distributed project teams. They were able to more effectively coordinate the development of software components for the module because the project deliverable was decomposed at the level of the required features and functionality. Project teams that benefitted the most from small releases were ones that collectively focused on the critical features first. That is, the critical features were distributed among developers at different sites. Through testing and continuous integration these features were incorporated into the next release. The

next batch of features and functionality would then be distributed across the different sites. Some of the less successful project teams did not systematically coordinate the small release effort. Instead various features and functionalities were assigned to the different sites without any explicit prioritization. This approach made it difficult to coordinate schedules in the face of impending deadlines for project deliverables.

Limitations As long as one of the project team sites was in proximity to the customer organization, spatial, temporal, and configurational dispersion did not present any major challenges to the ability to deploy small releases.

Practical Tips

Tip 1 Small releases can be an effective tool for coordinating developer work across time and space. The key to the effective implementation of this practice is to prioritize and distribute the development of the core features and functionality across the geographic sites involved. This will ensure that development efforts are focused on a core set of features and functionality at a time. This will also make it easier for developers at different sites to coordinate their schedules for delivering required functionality. This is much more difficult to orchestrate when priorities across sites are not aligned.

2.5 Conclusions

As noted at the beginning of this chapter, agile methods and distributed forms of organizing both hold great promise for enabling software project teams to meet the challenges posed by today's demanding software development environment. The purpose of this chapter was to provide some insight into the considerations involved in combining the use of agile methodologies with distributed forms of organizing software project teams. This was accomplished by

- (1) highlighting the different ways in which distributed teams can be dispersed, and
- (2) highlighting some of the successful and unsuccessful approaches taken by the software project teams that implemented XP practices.

From this discussion I developed recommendations about how teams should be structured in order to implement practices associated with one specific agile method—XP.

The discussion on implementation strategies clearly indicates that no single size fits all XP practices. Specifically, the challenges and opportunities associated with implementing each XP practice differ for each form of dispersion. The communication needs associated with implementing each of the XP practices also differ quite

markedly. Consequently, there is no single solution to optimize the implementation of all XP practices [3].

It is also clear that each practice differed in its overall function in the project team. Some practices served a planning role (e.g., planning game, collective ownership, coding standards) that facilitated the execution of action-oriented practices (e.g., pair programming, refactoring, continuous integration). This also meant that the effectiveness with which each practice could be implemented was affected, to some degree, by the overall role played by each practice.

The challenge for project managers is related to determining what is the most important in terms of the design of the team—i.e., how team members are dispersed across sites—and the specific XP practices that are germane for achieving team objectives. In some cases project managers may have little control over the team design, in which case decisions need to be made about which XP practices to implement given the existing team structure. In other cases certain XP practices may be needed in order to successfully achieve project objectives and project managers must make decisions about how to structure the team to execute those practices. Undoubtedly, this requires tradeoffs to be made. Project managers must weigh the costs and benefits of various options and pursue the course that optimizes the project team's chances of successfully meeting project objectives. The recommendations outlined in this chapter are intended to guide project managers in this decision-making process.

Acknowledgements I would like to thank Viswanath Venkatesh for his valuable input on earlier drafts of this chapter. Thanks also go to Jaime Newell for her assistance with formatting the chapter.

References

1. Abrahamsson, P., & Koskela, J. (2004). Extreme programming: A survey of empirical data from a controlled case study. In F. Juristo & F. Shull (Eds.), *Proceedings of the ACM-IEEE international symposium on empirical software engineering* (pp. 73–82). Redondo Beach: IEEE Computer Society Press.
2. Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., & Ó. Conchuir, E. (2005). A framework for considering opportunities and threats in distributed software development. In *Proceedings of the international workshop on distributed software development (DiSD)* (pp. 47–61). Vienna: Austrian Computer Society.
3. Batra, D. (2009). Modified agile practices for outsourced software projects. *Communications of the ACM*, 52(9), 143–148.
4. Conchuir, E. Ó., Ågerfalk, P. J., Olsson, H. H., & Fitzgerald, B. (2009). Global software development: Where are the benefits? *Communications of the ACM*, 52(8), 127–131.
5. Cramton, C. D. (2001). The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3), 346–371.
6. Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*, 15(2), 200–213.
7. Fowler, M., & Highsmith, J. (2001). Agile methodologists agree on something. *Software Development*, 9, 28–32.
8. Holmström, H., Fitzgerald, B., Ågerfalk, P. J., & Ó. Conchuir, E. (2006). Agile practices reduce distance in global software development. *Information Systems and Management*, 23(3), 7–18.

9. Leary, M., & Cummings, J. (2007). The spatial, temporal, and configurational characteristics of geographic dispersion in work teams. *MIS Quarterly*, 31(3), 433–452.
10. Maruping, L. M., Venkatesh, V., & Agarwal, R. (2009). A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research*, 20(3), 377–399.
11. Maruping, L. M., Zhang, X., & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, 18(4), 355–371.
12. Maznevski, M., & Chudoba, K. (2000). Bridging space over time: Global virtual team dynamics and effectiveness. *Organization Science*, 11(5), 473–492.
13. Murru, O., Deias, R., & Mugheddu, G. (2003). Assessing XP at a European Internet company. *IEEE Software*, 20(3), 37–43.

Further Reading

14. Beck, K. (2000). *Extreme programming explained*. Reading: Addison-Wesley.
15. Beck, K. (2003). *Test-driven development by example*. Reading: Addison-Wesley.
16. Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*, 18(2), 22–29.

Chapter 3

Transitioning from Distributed and Traditional to Distributed and Agile: An Experience Report

Daniel Wildt and Rafael Prikladnicki

Abstract Global companies that experienced extensive waterfall phased plans are trying to improve their existing processes to expedite team engagement. Agile methodologies have become an acceptable path to follow because it comprises project management as part of its practices. Agile practices have been used with the objective of simplifying project control through simple processes, easy to update documentation and higher team iteration over exhaustive documentation, focusing rather on team continuous improvement and aiming to add value to business processes. The purpose of this chapter is to describe the experience of a global multinational company on transitioning from distributed and traditional to distributed and agile. This company has development centers across North America, South America and Asia. This chapter covers challenges faced by the project teams of two pilot projects, including strengths of using agile practices in a globally distributed environment and practical recommendations for similar endeavors.

3.1 Introduction

In the last decade large investments have enabled the move from local to global markets in many business areas [3]. In the same period of time, the global software market has undergone several crises: not only a large number of project failures have plagued the industry, but also the increasing demand for new systems has been strongly affected by scarcity in appropriate competences. In such environment, Distributed Software Development—DSD—provides a feasible alternative [2, 14]. Distributed software development has been increasing during the past years [8, 14]. Organizations search for competitive advantage in terms of cost, quality and flexibility in software development, looking for productivity increases as well as risk

D. Wildt (✉) · R. Prikladnicki
Agile Methodologies User's Group (GUMA), Porto Alegre, BR, Brazil
e-mail: daniel@facensa.com.br

R. Prikladnicki
e-mail: rafaelp@puers.br

dilution [11]. Many times the search for these competitive advantages forces organizations to look for external solutions in other countries, identifying global software development strategies. The key existing challenges however are related to strategic issues, cultural issues, technical issues, and knowledge management [9]. Various companies are outsourcing to third party consultant companies situated in emergent countries, while others decided to open captive development centers with full-time employees in order to develop internal Information Technology assets (Internal Offshoring) [12, 14]. Organizations aim at the development of large projects in offshore centers (1+ year duration). This is because long projects are more suitable to absorb the offshore centers' learning curve regarding the domain knowledge [3].

Usually, an initial strategy defines that offshore centers would be responsible only for the development effort [3]. This scenario supports an adoption of waterfall approach as the standard lifecycle. Onshore teams are usually responsible for planning and testing efforts while offshore teams are primarily assigned to developing tasks. While captive offshore development centers are increasing their business domain background they continue to follow waterfall life cycle, and its exhaustive documentation, which, in fact, is recognized as larger than in similar co-located projects, can impact team's productivity negatively. Team members that have an acceptable business background start to struggle on processes and controls that do not make sense to their daily activity. Thus it is not acceptable to invest so much effort on documenting and creating controls as used on the initial strategy of the offshore centers.

In this context, we report the experience of transitioning from distributed and traditional to distributed and agile, commenting on two pilot projects within a global company that used to have waterfall as standard software development life cycle. We present challenges faced by the project teams, as well as the benefits of using agile practices in a distributed environment, the impact of changing the waterfall approach to iterative development, and practical recommendations for those starting similar endeavors.

3.2 Case Overview

This chapter is based on experiences from a multinational computer company with headquarters in the U.S. The company maintains software development centers in Americas and Asia. We contacted stakeholders in both the Brazil and U.S. locations. There were 500 people working with software development in Brazil, 1000 in India and around 500 people working in the IT department within U.S., as shown next.

Table 3.1 Case synopsis

Company: UShardware ^a	
Number of developers	2000 in total, 500 in Brazil, where the study was executed
When was agile introduced	2005
Domain	Hardware manufacturing

^aName changed due to confidentiality reasons

From the U.S. perspective, the company has one main business model for DSD: internal offshoring (both Brazilian and Indian centers develop internal projects for the headquarters' client located in U.S). Technical teams work only for captive offshore development centers. This company developed a set of global standard processes following three market standards:

- Capability Maturity Model Integration (CMMI)
- PM Body of Knowledge (PMBok)
- Microsoft Solutions Framework (MSF)

In order to disseminate software development culture and processes globally, this organization decided to pursue Capability Maturity Model Integration (CMMI) level of maturity assessment. Available set of processes were only based on waterfall life cycle. Project team members follow available processes and phases, regardless project size and complexity.

According to the organization strategy and a set of processes available at that time, documentation is generated onshore. Software product is developed offshore and tested back again onshore. In the past, distributed location justified extensive documentation while offshore centers were dealing with their domain learning curve acquiring business knowledge.

Nowadays this scenario has changed. Offshore centers have 5+ years of background on application domain knowledge. They are working on large, medium and small projects. In addition responsibilities have increased from "only development center" to a center responsible for planning projects, developing, testing and requirements engineering.

According to a set of defined processes in this organization project life cycle should follow extensive waterfall phases generating a significant amount of required documentation. Lessons learned showed that for larger projects (1+ year duration) standard practices were suitable. On the other hand, for medium and small projects (up to 9 months duration) standard practices were adding too much of complex bureaucracy to team members. In addition business was not satisfied with results achieved by the end of project as also captured on lessons learned sessions.

For medium and large projects documentation produced usually were outdated by the end of the project because business most of the time was maturing and changing their business requirements along the project. Due to this dynamic and changing environment some issues were listed as possible causes:

1. Powerful project managers were putting processes up front of business interests causing frustration on business expectation;
2. Weak project managers were accepting so many change requests flooding team work by paper work in order to keep documentation records up to date. Some projects had decided to have two project managers. One to track project status and another one only to maintain documentation. This practice was causing team members frustration because they were moved out from technical tasks.

Projects lessons learned sessions also showed that benefit of documentation was low in comparison to effort required to generate and maintain it. Often developers

with new assignments were contacted to resolve issues on past delivered projects. Application support teams instead of using existent documentation were reaching developers to resolve production issues. Those facts demonstrated that processes should be adapted to medium-small projects on this company. The organization decided to innovate on medium-small projects life cycle. It was clear that waterfall was not working for short-term projects. Agile methods were then selected to be run on two projects.

An assumption was that agile methods would bring more interaction between business partners and technical team. Moreover, unstable requirements would be gathered on iterative way along the project as business partners were expecting.

3.3 Transitioning to Agile in a Distributed Environment

This chapter shares experiences lived from 2007 to 2009. Looking at the adoption of agile methodologies in Brazil, a new team member joined the related portfolio, with previous experience adopting and adapting team realities. Looking at teams not using Agile Methodologies was just a motivation to start.

The Software Engineering Process Group (SEPG) was based on traditional development, globally. Some initiatives running in Brazil and UK were getting attention. Those two groups merged, and started to share experiences. The discussion moved to the SEPG, where a new Software Development Life Cycle was standardized: The Agile Development Method.

The designed agile process was based on eXtreme Programming. With discussions, Scrum and Lean started to grow and attract the attention in the process and a rich set of agile practices was put in place. Since the company was utilizing distributed teams, and running a CMMI level 3 based process, compliance with CMMI was always necessary and became a challenge on the path towards implementing agility. Nonetheless, the attitude inside teams changed with continuous delivery of value after each cycle of development and better organization inside teams. The company was then able to use Agile Methodologies and yet be compliant with the maturity model in place, using the best of both [7].

The company defined the work having three different levels of agile methodologies:

- Operational/Engineering: team members were able to set expectations with technical teams, while looking at eXtreme Programming practices,
- Tactical: business team and management, looking at Scrum,
- Strategic: the whole organization looking at Lean culture and Agile principles and values, with a continuous improvement mindset.

To help on engagement, evolution and adoption, a global community of practice started, grouping all related information about agile methodologies and looking inside the organization for other groups trying agile, bringing them to the same page. Discussion forums, white papers, online webcasts to share information across the

globe and local trainings and presentations were some of the initiatives brought together to help those teams looking for the transition.

Moreover, each site interested in the transition had local help from a Coach, to help understanding the environment and help prepare teams and customers for the transition. In the Brazilian site, one person helped other teams formally, with 20% of its time available to help other teams to adopt. The same person was responsible for local internal trainings on agile methodologies.

In every team, there was only one rule, to work with baby steps, trying one specific practice, perceiving its value, and then try a new one. All of this based on the cultural change needed to support the transition. The objective was simple. We wanted to fail as fast as possible, showing one of the benefits to work in an iterative and incremental development focused on customers, continuous improvement, and quality and team self-organization. In order to share the experience lived with the two projects, we first present a brief overview of each project. Due to confidentiality reasons, we will reference the projects as A and B.

All projects faced challenges while adapting existing organizational processes to agile practices. Table 3.2 summarizes the main challenges identified.

Table 3.2 Challenges identified

Challenge	Project	Phase
Lack of formal document for requirements	A	Planning
Inadequate team structure	B	Planning
Communication issues	A, B	Developing
Difficulties on estimating story points	A, B	Planning
Management not used to agile practices	A, B	Closing

During the explanation of projects, we will make it clear where these challenges were faced. The challenges reported were discussed during retrospective sections in the end of each project development cycle. All teams found ways to improve their processes, also helping them to increase their maturity while using agile principles and practices.

3.3.1 Don't Tell What Agile Is and Be Successful

Project A is a customer relationship domain project related to customer integration on manufacturing business processes. This project had 7 team members located in North and South America.

We changed the way the team was working just introducing agile practices and not telling the whole team what was really being applied. In this context, most of team members involved were not from Information Technology area and were focused on their tasks only. So being agile there was nothing but an adjective.

Table 3.3 Project overview

Project A	
Duration	5 months
Status	Finished (measured during 5 months) Focus on Scrum Practices
Agile practices	Daily meeting, Iteration Review, Iteration Retrospective, Customer Tests, Collective Ownership, Continuous Flow, Whole Team, Sustainable Pace
Locations	US, Brazil

Table 3.4 Project A: team

Locations	Number of members	Roles
US	2	Product owner
Brazil	5	Developer, Tester, ScrumMaster

The team located in Brazil was new to the customer integration area. We started using the available process, where each team member had a list of integration activities to work on, similar to a personal backlog. Every integration activity was based on EDI (Electronic Data Interchange) integrations to be built and validated between project team and customer team.

This project had a formal requirements document, but it was not standardized with the business team. This caused the team working in processes not ready to start or processes that were not real priority. Actually priorities were a real issue within the team. Most team members were multi-tasking, with no real focus on what's really important for business.

Consequently command and control management and micro management, bad management practices looking at agile principles, were the only way to make sure team members were working on what's important, since it was not clear what integration process need to stop to prioritize what's on the "main" priority list.

A lack of a standardized requirements document was a big issue and led to a large share of impediments during the project execution. Lots of activities were impeded because information was missing. And since feedback was not online, it took days to receive an update about an impediment. To avoid this, we reviewed the requirements document, which reminded more an epic user story than anything else. We made it clear that an activity could only be prioritized if we have the input document available and related documentation. Doing that, we guarantee a better input, and expect a reliable output for activities. The size of the cycle was not a problem at this moment, but the large amount of work in progress.

Next change was related to multi-tasking. Some team members had 20 activities "in progress". Actually two or three in progress and all others impeded. With individual to-do lists, it was impossible to manage priorities and check if next activity could be handled for someone free to start. In this situation, with self-organizing teams working with self-assigned tasks, and one prioritized list we could achieve

more focus from the team. The Brazilian team merged individual to-do lists into a big one. The U.S. team was responsible to prioritize the list based on business needs. We decided that priorities could change every week. It could be also changed earlier if it was something really important. Thus all team members in Brazil were able to work in any type of integration. We were not segmented anymore. Collective Ownership was in place.

The Brazilian team started a daily meeting using phone, and that meeting was open to the whole team. That meeting was an opportunity for people outside Brazil to check how work was being done and help on impediments raised during the short meeting. Meeting time was defined in an agreed time for those outside Brazil and for people in Brazil. Every team member had to answer three questions, (1) what I was doing since the last meeting, (2) what I'm going to do until the next meeting and (3) is there something blocking my work?

In addition to this, we had one meeting every Monday to make sure we were looking at the right priority list. The flow was continuous, so this team was not having the iteration concept. Every time one finishes an integration activity, he/she pulls the next one at the top priority. This meeting was a weekly cycle meeting, to get whole team attention and update/refresh priorities. This meeting took 1 hour every week.

We had another meeting in the end of the week to check improvement opportunities. That meeting ran using a retrospective meeting approach, where we set the stage and then use three different moments: (1) review the past two weeks looking at ups and downs, (2) brainstorm to find opportunities and requests for behavior change in the team, (3) vote and get three most voted items to prioritize in the next cycle. This meeting also took 1 hour every week.

With these changes, anxiety was overcome. The team had one backlog list to work on, the business was aware of current evolutions, and new priorities were clear and established with the whole team.

3.3.2 A Fully Cultural Transition from Traditional to Agile Development

Project B is related to outbound and fulfillment processes, focused on Global Processes with 20 team members in 7 different locations.

This project provides experience with integrating extreme programming and Scrum, in different locations, with people from different cultures, with different skills and a rich experience with other agile methodologies. There was a coach assigned for this project, helping the teams to understand practices such as daily meetings. Those took place first with the Brazilian team only and then the US team attended meetings together.

Later on, two daily meetings were implemented due to a high distribution of the project teams. One meeting involved the teams in UK, Russia and China. The other meeting involved people in Brazil, Canada, Mexico, US and UK. The Project

Table 3.5 Project overview

Project B	
Duration	14 months
Status	Ongoing (measured during 14 months) Focus on Lean, Scrum and XP Practices
Agile practices	Iteration planning, daily meeting, Iteration Review, Iteration Retrospective, Customer Tests, Collective Ownership, Small Releases, Whole Team, Sustainable Pace, Test Driven Development, Continuous Integration, Refactoring, Pairing, Coding Standards, Limited Work in Progress, Self-Organizing Teams
Locations	China, US, UK, Mexico, Canada, Russia, Brazil

Table 3.6 Project B : team

Locations	Number of members	Roles
US	2	Team lead (global), Product owner
Brazil	9	Developer, Tester
China	3	Product owner, Developer, Tester
UK	2	Product owner, ScrumMaster
Mexico	1	Product owner
Canada	1	Product owner
Russia	2	Developer, Tester

Manager was in the UK, and he was the “glue” between different time zones. This project was using a Scrum of Scrums [15] approach to keep communication and transparency.

Organizing the meetings were not the real problem in this project. Lack of formal requirements was identified as the barrier for the project teams that faced challenges in terms of capturing business requirements in an easy and effective way.

In the first approach, documents were large, and after meetings and meetings the business partners signed-off the requirements. Later on, however, defects showed up due to bad requirements gathering, even though the document was approved. Business partners were aware they could throw a change request or call it a defect, to get what they really need.

Project B had developed a software requirements document but experienced difficulties on having it signed off by business partners that were located in Asia. This fact caused a lack of commitment from business to agreed requirements during project development. After requirements were released for the final testing, business partners found defects instead of development teams and most defects were actually related to changes in the original requirements. This environment changed the process later on, as we will describe further.

Inadequate team structure can be captured as inappropriate decisions in terms of team organization and task assignment. For instance, this project faced a barrier

between generalist against specialist behavior in the technical team. Agile practices state that every team member must collaborate as a generalist in the project tasks. In this project even with performance testing training available test team avoided to execute performance testing because they were primarily functional testers. Root cause for this reaction is a common human avoidance of missing his/her knowledge silo and avoidance to accept changes. Other than that, testing teams were not accepting to test partial delivery of requirements, to help on validation and work on prevention of defects.

Every meeting was an opportunity to create knowledge and improve communication. That works for planning, daily meetings, reviews and also for estimation session. Teams had used planning poker as the main estimation technique, and the best example rely on Project B usage of the technique. Planning poker [5, 10] is a simple game used to estimate the work from a team perspective. It helps in knowledge creation and consensus creation by stimulating discussion about every feature being played. To start the game, a moderator needs a list of features. These features are written usually with a user stories approach and using a unit for sizing. Usually a team can start with T-Shirt size estimation, using small, medium and large, and find a category to fit that feature in. In the team's experiences, using a Fibonacci set with 1, 2, 3, 5, 8, 13 and "?" was enough to create these boundaries.

To have the game working, every local team defined a leader, to make sure everybody in the local team has a basic context before going online and have the discussion with a bigger group. One of the first activities planned was to have the team looking at features and checking if they were testable, if the business relevance was available and making sure they had the context. If something is not ok, a business user can be available to answer questions. The availability doesn't have to be online, it can be by the use of a tool to help on communication, like wikis or forums. So, in order to come into an estimating game, local teams were aware of what they would estimate and what would be on discussion. The remote and distribute exercise helped the whole team to understand features and create knowledge.

The main challenges for all teams were related to low participation in the game, and the fact that running the game locally the moderator may observe the team members behavior, which is unavailable most of the time when working in a distributed environment.

In project B a lack of team maturity in agile practices impacted team capacity evaluation. Instead of cutting a sprint scope and postponing the work till the following sprint, the testing team decided to test the outcome using the overtime to deliver the goal. That resulted in defects and test scenarios not mapped into acceptance criteria. Those decisions caused team rework and overtime generating frustration and impacting software quality in terms of defects. Also, the testing team was focused on post-development testing, in other words after the development team finished 100% of the features. That impacted the focus on prevention needed for testers inside the Agile Team. Without pairing with the development team, the testing team became a "reaction" team, just there to find defects, and not to avoid them beforehand.

How to change this? Better communication. Some developers started to pair with the testing team, to establish acceptance testing and create the ownership during

the development process. Developers knew how to develop some tests, and testers complemented building tests with the expertise they had.

Continuous integration received more attention too. The team was not only using a continuous integration server, but also added tools to help achieve the source quality. A source code audit, code coverage, code metrics were implemented to help the team to understand the next work tasks when idle during an iteration.

3.3.3 Benefits of Using Agile Methods in Distributed Environment

The organization faced many challenges in their endeavor of implementing agile since they were more familiar with the waterfall life cycle. Nonetheless, the lessons learned have also captured a list of important advantages from the application of agile practices and Scrum.

Bi-weekly software product builds were successfully used in project B: releasing as many builds as possible project team eliminated the waste in terms of waiting for a whole package to be tested [15] and get earlier feedback from the management and business units. Team used to release at least three times a week to get feedback from testing and business team. Every result could be an opportunity to use that technology being tested inside a running project. In project A, results were delivered weekly, since the need to review priorities was higher.

Even though communication with business partners sometimes required more time than expected to approve new requirements in project B, they participated and collaborated with the project team anticipating tests and providing adequate support during development and on defects tracking. A shorter development cycle was used reducing risks and increasing feedback for other teams. More communication was needed and it was achieved also because of the use of short cycles. Since time was shorter, more communication was needed to make sure next priorities in a product were being prioritized correctly.

Distributed teams were able to look for the same goals and work more integrated than usual. There was no time to review and wait decisions. Agile brings more attitude to the team, providing a pace of continuous delivery of working and valuable software.

Project A reported communication issues as one of the challenge faced. Adding the Scrum framework to the process gave the sustainable pace the team was longing for, since the same project also described strength that every sprint delivered not only increased the team motivation but also improved the collaboration and engagement. Comparing interaction and cooperation levels between the technical teams and business partners during the first sprint and the last sprint measured, a major improvement could be noticed in relation to the recorded lessons learned.

It is possible to use Scrum and other agile methodologies to minimize some of the main difficulties that distribution brings, such as the lack of communication, feeling of distance, lack of “teamness”, create synergy, responsibility and accountability, among other factors.

3.4 Practical Recommendations

Based on challenges and strengths faced, project teams were able to document a set of recommendations for projects within the same environment and scenario, which can be also of interest for the companies undertaking the similar endeavors.

Practical Tip: Use cases should be transformed in user stories with the focus on testing

Project B decided to change the way they used to document requirements and started to use a user story approach. User stories can be written having all information needed to support a requirement. Answering questions like what, for whom and why it is needed, helps in the basic understanding of a feature. Adding acceptance criteria for every user story will lead teams to think more about testing, while thinking about the given when and then clauses. Those are actually action and reaction clauses that can be found on use case documents, but in a standard language. The need of a standard way to write and maintain features with requirement documentation can be supported writing testable and executable documentations. What happened was that the development team started to write the user stories in a higher level, thinking more about the business. And another benefit was that the team was able to merge the business requirements, technical information and test scenarios in one document mapped from a pairing session with the development and the testing teams. Documents started to become leaner, and focused on the real needs.

Smooth Transition: start resuming current requirements into user stories approach with acceptance criteria, and increase its usage from one cycle to another or from one release to another.

Pitfalls: Do not change documentation 100% from night to day. Use one current release to start playing with a new way to document, and get approvals inside team. Validate and increase its use among other teams. Make it a team ownership, get suggestions and improve to make it feasible for use widely.

Practical Tip: While piloting agile practices the team must document Scrum iteration

This is necessary in order to have a better material to check on retrospectives. Mapping agreements, which are usually something to share on a retrospective, and follow up on impediments help teams to understand what happened during that time. What can be done is use a tracker (extreme programming role) to map things that happen during an iteration and discuss that later on a retrospective. Tracker can help team to maintain a burn down chart to measure flow of features that are adding value to the product, and can write a blog or something to tell a history about the iteration. With this, team will have historical data to understand what is happening. Tracker

role is something to rotate inside team. This way, this is one practice to help on leadership creation inside team. New leaders can grow and new ideas can be suggested, since someone different will be looking at team problems during iterations.

Pitfalls: do not document everything. Tracker should check highlights and lowlights of each iteration day, and focus on what is adding value to the product. Document errors and warnings to talk during retrospectives and see how those can be avoided.

Practical Tip: Continuous training sessions to share and remember practices, principles and values

These are important to keep team discipline and to add new practices the team decides to focus. A Coach or ScrumMaster must be helping the team to solve impediments and support discipline during one iteration, training and pairing with the team when needed. Looking at technical side, practices like a coding dojo [4] could be used to help teams to improve test automation capacity and to share knowledge about different tools. The same can happen for user story writing, testing and estimation.

Smooth Transition: find a specialist in one programming language used by team and propose a challenge to be coded using test driven development [1], refactoring and other eXtreme Programming practices. Do the same thing for User Story writing, creating the opportunity to have more people understanding how to create user stories. As soon they like to do that in Dojos, they will like to do that within projects.

Practical Tip: ScrumMaster needs to be a strong negotiator

And also owns a *de-facto* political power over technical team and business partners. Since agile practices are based on team collaboration and trust, a ScrumMaster needs to be a person strong enough to deal with business in terms of requirements priority, problem solving, and also to work with team members to deliver on agreed time box. In a global scenario, this is even more complex. For this reason, the ScrumMaster needs to combine not only leadership on project team and business partners, but also cope with distance, trust, and cultural diversity. But, do not rely only on ScrumMaster. Every team member need to pull responsibility and ownership, looking for transparency always.

Smooth Transition: using a tracker and doing Coding Dojos are ways to increase leadership.

Practical Tip: Assign testers and developers to work together

This is an important recommendation, achieved in Project B. For instance, assigning testers to peer review unit test scripts can improve unit tests coverage. This

decision can help on anticipating defects to development phase instead of finding only after promoting build to test team. In project B, developers and testers worked in silos, impacting team collaboration. Moreover, dislocation between engineers promoted an environment where each group created their own practices, impacting code quality. Every pairing activity was good enough to add value and help on knowledge creation between development, testing and business team. That became a usual practice for project B and helped team to increase business knowledge and increase team spirit. Testing team started to be a prevention team, not a reaction. Also, doing this makes clear that it's about only one team, not development and testing team. It's just one team.

Smooth Transition: pairing is a good practice to create knowledge. Create opportunities to create knowledge and share. When creating knowledge, pair.

Practical Tip: Increase testing practices, automated if possible

This means the use eXtreme Programming [1] practices to increase the use of automated tests and practices like continuous integration, that are excellent to support distributed teams and guarantee that the software codebase is correct and with integrity. Projects B and C focused on adding this to their map of practices used. While Scrum can be useful to support management and bring problems to surface, when a problem is related to software engineering, the use of extreme programming practices can add value. The use of static audit tools and code coverage tools can help the team to keep track of source code and code quality.

Smooth Transition: automate acceptance criteria, use coding dojo to increase team skills for testing.

Practical Tip: Keeping valuable documentation

Moving project B from a waterfall to an iterative and agile approach did not change the need for documentation. Since the team is working in a distributed environment, the documentation may even increase [6], and the team will need to adapt some of the documentation to become more effective. For this reason, one should define and document what brings value to the project and what can be useful to the distributed teams. Inside project B, documentation become more focused and only important and mandatory documents continued to be written.

Smooth Transition: question documentation in every continuous improvement cycle. While doing this, suggest changes and show how it would look like to others.

Practical Tip: The use of a “global” taskboard

This kind of tool can help on improving the productivity of global agile teams. The use of applications to share knowledge, like wikis or file sharing, helped teams

to control the activities planned for each iteration and the product backlog. Even using spreadsheet was valuable, since every team member could edit and share evolutions with the rest of team. This sometimes caused delays and unexpected problems, but that was also a way to keep discipline to share status with teams in different time zones. A tool to help global teams on the planning and execution of a sprint is a good approach to manage visual management.

Pitfall: Task board should be updated by the team, not by one member or project manager. Team owns the task board.

Practical Tip: It's important to have multi-disciplinary team running the planning poker game

In a planning poker session different perspectives can help building the shared view. So developers, testers, database administrators, project managers, business analysts, all roles in a project are welcome to help in one of those sessions. The integration with technical roles and business roles are important to increase success rates. And in a distributed environment this is even harder to achieve.

Smooth Transition: use three different roles while estimating, like developer, tester and analyst for instance.

Practical Tip: To have the planning poker game working in a distributed environment, every local team needs to have a leader

The leader will make sure that everybody in the local team has a basic context before going online and have the discussion with a bigger group. The remote and distribute exercise will help the whole team to understand features and create knowledge.

Smooth Transition: for every estimation game, define one person responsible to be the local leader. This way, team will increase business knowledge.

Practical Tip: Leadership is needed, not only leaders

It is important to grow leaders inside teams, and not expect ScrumMaster, Coaches or Project Managers to take all responsibility. They are part of the team, they do not own the team. So every team member need to understand they have a active voice inside the process.

Smooth Transition: use different local leaders for estimation games, use trackers to understand an iteration, coding dojo moderators, as ways to create and increase leadership.

Practical Tip: If you need one practice and one value. . .

Value communication and practice continuous improvement. Find a spot inside team's agenda to have a retrospective meeting. Find improvement opportunities and things to work on prevention. Use whole team capacity to create these opportunities, with techniques like brainstorming, mind maps, and other practices to share knowledge and raise a common understanding about a problem.

Pitfalls: do not think that what other team is doing is what you need to do. Value what you have currently and check with team members what they want to improve, what they should change, in order to have more suggestions.

3.5 Conclusions

In this chapter we have presented the experience on the usage of agile practices within a company that has globally distributed project team members. Particularly we discussed the challenges in implementing Scrum by the company whose culture was not agile-oriented, and thus it required a change of the mindset for people to see and experience the advantages of applying Scrum together with traditional practices already in place. Despite the initial skepticism, the results were very positive—it provided a great opportunity to better integrate distributed team members and bridge them with the business units. In addition, the experiences with the two projects showed that it is also possible to use Scrum and other agile methodologies to minimize some of the main difficulties that distribution brings, such as the lack of communication, feeling of distance, lack of “teamness”, among other factors. The lessons learned suggest that Scrum complemented with other agile methodologies like eXtreme Programming become very important to increase overall team abilities and the software product quality, looking from the source code perspective.

References

1. Beck, K. (2005). *Extreme programming explained: Embrace change*.
2. Bohem, B. (2006). A view of 20th and 21st century software engineering. In *Proceedings of the 28th international conference on software engineering (ICSE)*, Shanghai.
3. Carmel, E., & Tjia, P. (2005). *Offshoring information technology: Sourcing and outsourcing to a global workforce*. Cambridge: Cambridge University Press.
4. Coding Dojo. Available online. <http://codingdojo.org/>.
5. Cohn, M. (2005). *Agile estimating and planning (Robert C. Martin Series)*. Englewood Cliffs: Prentice Hall PTR.
6. Fowler, M. (2008). Using an agile software process with offshore development. Available online. www.martinfowler.com/articles/agileOffshore.html.
7. Glazer, H., Dalton, J., Anderson, D., Konrad, M. D., & Shrum, S. (2008). CMMI or agile: Why not embrace both! Available online. <http://www.sei.cmu.edu/library/abstracts/reports/08tn003.cfm>.

8. Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *Proceedings of the 29th international conference on software engineering (ICSE)* (pp. 188–198). Minneapolis, USA.
9. Herbsleb, J. D., & Moitra, D. (2001). Guest Editors' introduction: Global software development. *IEEE Software*, 18(2), 16–20.
10. Online planning poker. <http://www.planningpoker.com>.
11. Prikladnicki, R., Audy, J. L. N., & Evaristo, R. (2006). A reference model for global software development: Findings from a case study. In *Proceedings of the int. conf. on global software engineering (ICGSE)*, Florianopolis, Brazil.
12. Prikladnicki, R., Audy, J. L. N., Damian, D., & Oliveira, T. C. (2007). Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring. In *Proceedings of the 2nd int. conf. on global software engineering (ICGSE)* (pp. 262–271), Munich, Germany. Los Alamitos: IEEE Computer Society Press.
13. Robinson, M., & Kalakota, R. (2004). *Offshore outsourcing: Business models, ROI and best practices*. Alpharetta: Mivar Press.
14. Sengupta, B., Chandra, S., & Sinha, V. (2006). A research agenda for distributed software development. In *Proceedings of the 28th international conference on software engineering (ICSE)* (pp. 731–740). Shanghai.
15. Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed Scrum: Agile project management with outsourced development teams. In *Proceedings of the Hawaii int. conf. on system sciences (HICSS)* (p. 274). Washington: IEEE Computer Society.

Chapter 4

Tailoring Agility: Promiscuous Pair Story Authoring and Value Calculation

Steve Tendon

Abstract This chapter describes how a multi-national software organization created a business plan involving business units from eight countries that followed an agile way, after two previously failed attempts with traditional approaches. The case is told by the consultant who initiated implementation of agility into requirements gathering, estimation and planning processes in an international setting. The agile approach was inspired by XP, but then tailored to meet the peculiar requirements. Two innovations were critical. The first innovation was promiscuous pair story authoring, where user stories were written by two people (similarly to pair programming), and the pairing changed very often (as frequently as every 15–20 minutes) to achieve promiscuity and cater for diverse point of views. The second innovation was an economic value evaluation (and not the cost) which was attributed to stories. Continuous recalculation of the financial value of the stories allowed to assess the projects financial return. In this case implementation of agility in the international context allowed the involved team members to reach consensus and unanimity of decisions, vision and purpose.

4.1 Introduction

While agile approaches stress the fact that co-location is of essence, most distributed software development projects face the challenges associated with the distance when the engineering teams reside in different locations. Switching to one-site team is not always the answer, as projects strive for acquiring different domain expertise that can be often found only through a synergy of multiple collaborating partners. The case presented in this chapter describes how the need to cope with the distribution of domain expertise across teams from eight countries led to the tailoring and innovating of processes related to requirements gathering, estimation and planning.

S. Tendon (✉)
Agiliter Consultancy, Limassol, Cyprus
e-mail: steve.tendon@gmail.com

The implementation of agility suggested that all domain experts had to meet together to perform the planning activities. I asked the organisation to allow not only the product managers, but also a number of other professionals to work together for almost five weeks. The primary, grounded objection, was that these professionals could not be “away” from their offices for so long time. I suggested to let the team meet every other week, and asked the activities to be continued even while the team members were back at home.

As explained later, this was one of the best decisions taken in this project. The alternating of on-and off-site weeks not only enabled the people to attend to their ordinary duties with acceptable regularity (as to counter the original objection), but also allowed a very high level of involvement and feedback of other people in the home offices. This created the “ambassador” effect, which was a great motivator for the people who were sent to the off-site meetings.

The project had been given very little time to deliver results. The idea of promiscuous pair story authoring was prompted by the need to achieve results quickly, and to produce collective acceptance of the plan. The “one-on-one” discussion that took place during the pair authoring sessions were instrumental in developing agreement. It is easier to converge towards common conclusions when exchanging ideas and talking to a single person, rather than in a group setting. The pair authoring simply amplified the effect. The promiscuity (frequent changing of the pairs), made the convergence on any given topic spread across the team very quick, so that the team as whole would agree.

The focus on the economic value of a story point was instrumental to overcome distance and misunderstanding. First, it allowed to reason about the value of user stories with a metric that is common to all businesses: profit and return on investment. Second, it allowed to perform story triaging and avoid feature creep without upsetting any party involved. Agile proponents often claim that the agile methods are all about “adding value” to the client’s business. An essential aspect is how to measure such “value.” Unfortunately, most agile approaches give just face value to business value (pardon the unintended pun), but they have a hard time providing real numbers. The lack of financial metrics is the Achilles’ heel of agile. In the case described here, another important tool is added to the arsenal: the definition and successive re-computation of a story point’s economic value. From a distributed perspective, the importance of this is that its meaning is universal, and can traverse and be understood across organisational and national boundaries.

4.2 The Case

4.2.1 Background

This chapter is based on a case study in a multinational company (20.000 employees) providing information, tools and solutions to professionals in the Health, Tax, Accounting, Corporate Services, Financial Services, Legal and Regulatory markets, needed to improve the software development operations throughout their European organisation.

Table 4.1 Case synopsisCompany: NLDlegal^a

Number of developers	56
When was agile introduced	2006
Domain	Software for law firms and legal professionals
Project	Requirements gathering, estimation and planning for developing a European software business plan
Duration	5 weeks
Status	Finished
Agile practices	Tailored from XP, IFM and others, plus innovations
Locations	Belgium, France Germany, Italy, Poland, Spain, Sweden, The Netherlands and others

^aName changed due to confidentiality

The company had acquired a number of smaller software publishers during the first half of the '00s. The acquired companies were spread throughout Europe; had very diverse software engineering capabilities; used entirely different software technologies; targeted a number of different platforms; and used entirely different software development methodologies. Some of the acquired companies were supported by just one or two engineers (“garage bands”) and did not use any specific methodology. Others had tens of engineers, and used more formal methodologies, albeit home grown ones.

Several of the acquired companies developed the same kind of software, but tailored at the specific needs—in particular legal requirements—of their respective countries. A lot of engineering effort was duplicated in the various countries, and the company was particularly concerned about consolidating software development operations. The problem was: how to integrate 23 software business units that originated from 32 acquisitions over 11 countries with over 300 software engineers in total. The company had tried a number of times to achieve consolidation through various initiatives. Some attempts focused on technology alone (selecting development environments and deployment platforms). Others focused on organisational aspects. None of those initiatives were successful at introducing significant and enduring changes—the main reason being that there were too many walled gardens, silos and diverse organisational cultures.

In the period 2004–2005 I was called in as a software strategy, process and methodology consultant to assist in formulating an overall strategic software consolidation, process improvement and growth plan. Despite being a pan-European organisation with so many software business units, there were no distributed or multi-site software development projects taking place. One of the major propositions of the plan was to take up distributed, multi-site development. This was a natural consequence of having a consolidated software engineering capability, that nonetheless had to take into account the differences in culture, languages, customs and laws in the various markets targeted by the company. Such local customisations could best

be addressed by smaller local software engineering teams; while the generic application platforms and frameworks could be developed by larger, technology focused, co-located teams. In November 2005 I presented the strategic plan to the company's CEO and other C-level executives. The plan contemplated, among other things: software product line engineering, outsourcing and near-shoring, proprietary software architecture and framework construction, and methodology and process improvement initiatives.

One recurring theme throughout the proposal was the need to embrace agility, at all levels of the company, including the Executive Management. Agile was a new concept for them, but eventually, at the beginning of 2006, they approved the initial pilot project, in order to evaluate the viability of the agile approach.

4.2.2 Management Support and Sponsorship

In fact, this case begins prior to the approval of the pilot project, with engaging the Executive Management in accepting agility as an alternative approach. The reasons supporting agile had to be expressed in terms that made sense from a business management point of view (rather than a project management or technical point of view). Agile was justified on the grounds that software ventures require:

- A new style of “empirical” management where fiscal responsibility is exercised differently.
- An investment perspective that is not like a standard capital investment (e.g. “buying equipment”), but rather like funding a new venture (e.g. “buying knowledge”).
- An investment model that is similar to the one used by Venture Capital companies.
- Flexibility because full qualification of costs and benefits at the start cannot be expected.

I finally gained Executive Management's support and sponsorship when adoption of a “*discovery-driven planning*” approach was suggested as described in [1] for controlling all new software projects. It was understood and accepted that funding had to become iterative and incremental, so that it could incorporate the possibility to “*reconceive*” as explained by [2].

Practical Tip: When an organisation is large enough to face the challenge of distribution, the adoption of agility is not only a team decision. Often, several departments and business units are involved. For this reason it is essential to get management's understanding, support, buy-in, sponsorship and explicit endorsement: it must become a company objective to adopt agile.

4.2.3 The Pilot Project

4.2.3.1 Project Selection

The agile pilot was going to support a consolidation initiative in a specific market: the Legal market (lawyers, attorneys, law firms and legal professionals in general). This market was chosen for the following reasons:

1. The market was considered the one with the lowest risk in case of project failure (with respect to the other markets where the company was operating).
2. It had (in relative terms) the smallest development organisations, with 56 software engineers in total—including both R&D and maintenance resources.
3. It was representative of the multi-national setting: it covered eight different countries to start with (Belgium, France, Germany, Italy, Poland, Spain, Sweden and The Netherlands), with others (Czech Republic, Denmark and UK) to follow later.
4. It had already been the objective of two prior attempts at technical consolidation, and therefore had more detail information available.

4.2.3.2 Project Overview

The organisation served the Legal market with more than one application per country, for a total of 20 applications. Most of those applications were approaching the end of their useful life-cycle, and in need of redevelopment in order to keep up with new technology and evolving market demands. The goal was to ultimately deploy one application, supported by one single software architecture for the whole Legal market in Europe, and replacing all products of all countries. The objectives included:

- preventing local green field redevelopment;
- impeding duplication of efforts;
- avoiding local investments in sub-scale markets;
- realising a single cost base replacing multiple cost bases;
- reducing full time employee head count; and
- driving up ROS (return on sales) on a European level.

The project had to produce a business plan to allow Executive Management to perform an investment appraisal, and decide whether or not to undertake any further engineering and development efforts.

Notably, the main reason why the prior attempts did not achieve management approval was mainly of financial nature. The analyses that were produced were not showing satisfactory financial returns. The failure of these prior attempts was one of the motives that induced Executive Management to try the agile approach. If the agile approach could produce a convincing business case, then Executive Management could draw wider strategic conclusion about the validity of the different approaches.

Practical Tip: When transitioning to agile, a successful strategy is to pick the toughest project that traditional approaches failed to deliver and which recognizes the criticality for the change. (If the traditional approaches did not fail, you have no reason to move!) Set up a focused “SWAT” team, and let it crack the tough nut. Aim at gaining success. Show that agile can work in your context, through solving the most complex problems which other approaches could not solve. After the first success: rinse and repeat! Tackle the next toughest problem. It is hard to argue with success where you had failure before. And should you fail, you won’t get compared and measured against a successful precedent—which would be the case if you picked an “ordinary,” “bread-and-butter,” “easy” problem!

4.2.3.3 Participant Involvement

I asked the company to make available product managers, project managers, domain experts, accountants, software engineers and other professionals from the various countries, in order to execute the activities needed to produce the project plan. The activities were going to last for almost five weeks. The idea was to use XP’s collective roles of “customer team” and “programmer team”—or, as they were called in the project, the *Product Management Team* and the *Software Engineering Team*. The PM Team was going to identify—and agree upon—a single and common set of requirements satisfying the needs of all European countries. The SE Team was going to evaluate the feasibility of the requirements and provide estimates.

I asked every country to send four people to staff the two teams: an actual product manager and a domain expert (or, alternatively, a project manager) for the PM Team; and then a senior and a junior engineer for the SE Team. There were exceptions to this pattern, depending on the relative size of the countries. In any case, and in order not to unbalance representativeness, each country was requested to send at least one but no more than three representatives to staff any of the two teams.

The planning process covered a period of four weeks. Additionally, before the activities started, I held an initial three day induction workshop. Hence, the planning process was broken down in the five stages, as shown in the following table:

Table 4.2 Planning stages

Stage	Activity	Duration	Colocation
1	Agile induction workshop	3 days	YES
2	Initial requirements gathering with story cards	1 week	NO
3	Promiscuous pair story authoring and estimation with story points	1 week	YES
4	Time estimates, story revision and benefit/penalty ranking	1 week	NO
5	Financially focused planning game	1 week	YES

The initial workshop, the third and the fifth weeks saw the team member get together. Each time, a different city was chosen for the meeting—to even out the burden of travelling amongst all team members (who came from all over Europe). During the second and the fourth week the teams worked in their respective home offices. This alternating of on- and off-site activities was fundamental in the overall process; and particularly significant in order to liaise the team’s work with the respective members’ home offices. I moderated and supervised all activities that were conducted in co-location; and I gave instructions about what the teams had to do when working at home.

Practical Tip: When transitioning to agile, make it clear to everybody involved—from Executive Managers to the junior programmers—that it is not an easy “*let’s try and see*” approach. Instead, the entire organisation needs to commit to and support the transitioning. Resources have to be allocated, time schedules planned, and so on; at the same time as the daily operational activities are to be performed.

4.2.4 The Journey of Implementing Agility

4.2.4.1 Agile Induction (Week 1)

It was necessary to have an introductory workshop to explain the agile approach in detail. During the workshop, the teams could appreciate the iterative nature of the agile approach. They understood that even what would have been qualified as the requirements elicitation phase in a traditional setting, was going to be carried out in an iterative and interactive way. Furthermore, it was going to be a team effort, with all participants collaborating, rather than having some business analyst interview the single product managers, and compile requirement documents. The PM Team learned how to use Story Cards [3] to represent requirements. Similarly, the SE Team learned how to express story estimates with story points.

The teams spent a lot of time on getting ready to work together. Not only did they familiarise themselves with the agile approach, but they also understood how to attain participatory decision making as described by [4]. All team member gained an insight into how they could overcome different points of views, not by seeking compromises and face-value consensus, but by actively looking for win-win solutions. In particular, the teams learned:

- to allow for the full participation of all members;
- to strive for gaining a deep mutual understanding of each other’s needs and points of views;
- to elaborate inclusive solutions that contemplate everybody’s requirements; and
- to accept sharing the responsibility for all decisions taken.

The teams quickly went through the “forming, storming and norming” phases of the team development process. They were ready for the “performing” that they needed in the week after. At the end of the workshop, they knew how to act together to reach convergence and agreement.

4.2.4.2 Initial Requirements and Story Cards (Week 2)

The second week was a preparation week. The product managers worked in their own countries, at their offices. All product managers were senior and very experienced in their respective professional role, but they were new to agile. They had to “trawl” for requirements and express them according to the story card format “As a [type of user], I want [some goal] so that [some reason]” as described by [5]. I urged them to actually write the stories on physical paper cards—although most of them resorted to word processors, nonetheless.

They were first asked to get requirements, information and feedback:

- directly from end users;
- from their companies’ support organisations;
- from the sales and marketing teams;
- from the engineering teams;
- and also by examining the offerings of the competition.

They were not to limit, prioritise or in any way triage the stories—they had to treat this like a brainstorming exercise, where “anything goes.”

Considering that the companies had all dealt with requirements gathering in the most diverse manners, this preparation phase had the significant result of collecting all requirements from all parties involved and represent them in a common format—the Story Card format. This was the first step towards achieving some sort of commonality across the country offices. A simple step, but significant in the consequences that would become tangible later, in the following week.

Another result was raising awareness back in the “home” offices: involvement of local colleagues was high, since all wanted “their” ideas represented in the overall international project. All countries wished that “their” requirements would be given priority, and hence participation was intense—unlike when product managers alone had been interviewed by business analysts (earlier, when the traditional methods had been used). The active role of the product managers, and the involvement of all stakeholders in the local offices was pivotal, to build awareness, consensus and buy-in at “home.”

Practical Tip: When your project stakeholders are in different departments or business units, keep in mind that they might have conflicting objectives, and that they might want to compete for the project’s resources. Turn their concerns into a positive force for the project, and find ways to seek collaborative solutions, rather than competitive ones.

4.2.4.3 Promiscuous Pair Story Authoring (Week 3)

One of the major challenges was convincing all product managers to accept the opportunity, necessity and viability of developing a single plan for all of business units in all countries. Initially, there was a strong scepticism about the possibility of finding common grounds amongst all countries. The scepticism could be explained by a number of factors. Some were obvious, like differences in culture. Others were more subtle and unstated. For instance, at the outset country product managers were sincerely convinced that their own requirements were “special” and couldn’t possibly be shared by others. Another forceful factor was the defensive reactions that the proposed common project inevitably arose: the country representatives would naturally be pessimistic towards a project that was meant to replace their own familiar businesses, and possibly reduce local resources. Finally, one of the strongest reasons for doubting about the project chances of success was the fact that the two earlier, traditional, attempts had failed.

All of this scepticism was overcome during the third week, when I introduced the promiscuous pair story authoring activity. The team came together, in an off-site location. I gave the PM Team members the assignment of creating a unified story board of all their respective stories. A commercial web based agile project management tool was used. Anybody could add stories, as well as edit and modify any story created by anybody else. This rule was inspired by XP’s practice of collective code ownership, where any developer can work on any part of the code base [6]. The collective ownership was now applied to User Stories, rather than to code. The idea was that collective ownership would allow for convergence to emerge during the authoring process.

Through the web based tool the teams could easily interact with stakeholders in their respective home offices. Conversely, the tool allowed the teams to continue to collaborate even during the following week when they all headed back home to their countries.

The PM Team had to identify and agree on a common set of “*personas*” [5].¹ Any of their individually authored stories had to be related to one specific persona, which they had to identify on a common basis. Again, the participatory decision making exercises turned out to be instrumental, and allowed the team to quickly identify their set of personas.

After having adopted a common format for the user stories, the identification of a common set of personas was the second significant achievement in terms of commonality acknowledged by all country representatives.

The Innovation of Promiscuous Pair Story Authoring Representing requirements with User Stories is a common practice in agile methods. [3] describes in detail how to employ User Stories. According to Cohn, anybody on the customer team

¹A persona is a fictitious character, often with a real name and even personality, that is the protagonist of a user “story”—it helps visualizing the needs of a user, relating those needs to real-world people, rather than to abstract roles.

can write a story, and he defines the customer team as anybody who can “*ensure that the software will meet the need of its intended users. This may include testers, a product manager, real users and interaction designers.*” In all cases though, the actual writing of a story is the activity of a single individual.

The significant process innovation that allowed the team to jump right into the “performing” phase, and overcome everybody’s original scepticism, was to introduce *pair story authoring*. I mandated that *all stories* being added to the web based project management tool *had to be written by a pair*. The idea of pair story authoring, likewise the idea of collective story ownership, was inspired by XP, where the concept of pair programming is documented by [6]. The difference being, of course, that in this case the pairing was applied to the authoring of stories rather than to their programming. The pair story authoring concept was further inspired by how the pair programming was enriched by [7] and [8]. Belshee augmented the pair programming practice with the concepts of:

- knowledge cross-pollination,
- promiscuous pairing,
- least qualified implementer, and
- exploitation of the beginner’s mindset.

I brought over these concepts to this new setting of pair story authoring. When a new story was first being worked upon, the participant proposing the story was not allowed to transcribe the original story from the card into the system—or worse, copy and paste from a word processing document. The second pair member, typically one coming from another country, had to type the story into the web based project management tool. According to [6] and [3] a story card is a token of conversation. The proposing member had to converse and explain, verbally, the concepts represented by the story to the other pair member who was typing at the keyboard. Since, as it will be described shortly, the pairing changed very frequently, the original author had a very short period of time to make sure all concepts were effectively transferred to his pairing partner. While this can seem unworkable, in practice it turned out to be one of the most effective tactics that allowed the team to gain momentum.

To promote rapid knowledge sharing, the pairings changed and rotated very frequently. This attained *promiscuous pairing*, where each story was handled and authored by more than one pair. This caused *cross-pollination*, where knowledge was transferred from person to person, through direct conversation and interaction. Pairs were changed as frequently as every 15–20 minutes. The *least qualified* pair member was always the one who materially had to type the story at the keyboard. Anytime a pair changed, it was the newcomer to the story under consideration that was given the task of typing the information into the system. At the same time, the previous writer took over the role of the expert member, because she/he had been exposed to the story for longer—and it became her/his turn to explain the concepts to the newcomer. While requiring that the least qualified team member had to materially write the story can seem an expediency, it had the important effect of exploiting the *beginner’s mindset*. As explained by Belshee, this favours knowledge sharing, because the beginner is more receptive to absorb new knowledge very quickly.

As mentioned earlier, but worthwhile repeating because of the major effect observed in terms of effective knowledge sharing, anytime anybody proposed a new story (from her/his own set of stories), she/he was not allowed to input the story into the system. Instead, the other component of the pair, the one who was unfamiliar with the story, had to do the actual writing. The most tangible consequence of this rule was that it forced the original author to thoroughly and quickly explain her/his points to the writer, making sure to be understood. The short time imposed on this activity (time until the next change of pairing), had the secondary effect that focus was always kept sharply on communicating the most important and essential features of the stories, thereby enormously clarifying the verbalisation of the original requirement. Unnecessary or low-level details were left out as a side-effect of the time-boxing; all explanations were brief, essential and to the point. Productivity was high; a lot of work was done in a very short time.

Practical Tip: If you decide to use the technique of promiscuous pair story authoring, be sure to impose the time box (e.g. 15–20 minutes). At the beginning it can be hard: some participants might complain and say that the time is too short for them to explain all the details of their stories. Suggest that they might split those stories into several shorter stories; but do not concede more time than what you decided for the time box. Soon the effects of time boxing will be tangible. If you have eight pairs each working on a different story, the analytical productivity will be much higher than discussing a single story amongst 16 people; and every single discussion will be much shorter and to the point. The frequent change of pairings will increase the analytical power and productivity even more.

Approximately every two hours, I interrupted the story pair-authoring activity. The team members then worked as a group on the stories that had been authored during the previous two hours. Each story was displayed (with a screen projection off the web application), and presented by its original author. At that point each story had almost certainly been seen and worked on by most (if not all) members of the team. This step served as a collective “reality check,” and as an approval step of those stories. The process was quick and swift. Many conversations about each story had already occurred pair-wise, and a common understanding had already developed between all team members. Rarely did any story need further, deeper discussions.

Once all stories of the 2-hour work block had been examined and approved, they were passed on to the SE Team for estimation.

Practical Tip: Since this “reality check” seldom introduces further changes, you might be tempted to forget it altogether: don’t do so! The purpose of the reality check is two fold. First it does, indeed, allow the team to collectively approve of their stories. Second—and this is more important—it enforces the

team sentiment, and reassures the team they are all aligned and pulling in the same direction.

Estimation with Both Story Points and Time Estimates While the PM Team continued with promiscuous story authoring with another 2-hour work block, the SE Team took over the stories of the previous 2-hour block, and started estimating them. On a rotation basis, one representative of the PM Team joined the SE Team to present the stories that needed to be estimated, one after the other. The representative actively participated in the estimation process, by conducting and moderating the meeting (as he/she had learnt to do during the induction workshop). The representative also answered any questions that the SE Team might have had about any story that was under consideration. The SE Team used a wide-band Delphi “*Planning Poker*” estimating technique with a scaled Fibonacci sequence as described in [3, 5]. The story points used were ideal effort points; and in no way related to time estimates. Any story that got the *unknown* mark, was sent back to the PM Team for further elaboration—the outcome of which was either the story’s dismissal or the splitting up into two or more simpler stories. Any story that was sent back to the PM Team was then expedited through the next 2-hour block. Occasionally, even the engineering team members proposed new ideas, which were captured by the PM Team representative, and then brought back to the PM Team and subjected to the normal promiscuous pair-authoring process.

4.2.4.4 Revision and Benefit/Penalty Ranking (Week 4)

The fourth week was back in the respective country offices. The PM Team had now a complete set of stories which they had all agreed upon as representing the totality of requirements. They had one week to present and discuss the full story board with whomever was concerned by the project in their countries.

The purpose was to see if there was any gross oversight, and to verify that the stories could be “welcomed” back at home. If any new stories were needed, they had to be collected and subjected to the same process as before (the week after, when all team members would get together again). Surprisingly, very few new stories came out of this stage, confirming the quality of the work done during the previous weeks.

The PM Team also had to make the first attempt at coming up with a prioritisation for all stories. The product managers had to autonomously express a Benefit/Penalty ranking for all stories. This was done with a simplified version of what is described in [9] and [10]. This ranking was going to be used as input in the subsequent step (the following week) when the entire team was going to meet again.

Traditional Time Estimates In the meantime, the SE Team members also worked at their country offices. They had to re-estimate all the stories, but this time according to their traditional way, and express the estimates in ideal time (*man-days*). Each

country was using their own estimating techniques, based on their own experiences, with their resources and particular tools and development environments. Like for the PM Team, these estimates were going to be used as input in the subsequent step the week after.

The SE Team members had to give ideal time estimates because it was necessary to have some kind of reference “velocity” to be used in the planning process. Since it was the first time the approach was used by the company, there was no historic data to refer to. The mean of the estimated times of all teams was attributed to each story. The sum of those means gave an estimated project duration; with which a hypothetical velocity could be computed, simply as the ratio between total number of story points over the estimated duration.

Practical Tip: When transitioning to a new method, try to find ways to bridge the old practices to the new ones. This will increase confidence in moving to the new ones, as they can be interpreted in the light of experience.

Both the benefit/penalty rankings and the time estimates were collected into a spreadsheet. All countries then sent a copy of their spreadsheet to a coordinator that compiled a consolidated master spreadsheet out of all data, and computed the appropriate mean values of the benefit/penalty rankings and of the time estimates for each story.

4.2.4.5 Economic Value Calculation (Week 5)

The last week saw all team members reconvene, with the purpose of performing a *Planning Game*, but with a few variations on the classical Planning Game of XP [6]. The SE Team used the hypothetical velocity to check that the project (as a whole) was realistically sustainable by any “average” developer on “their” home team. After a few adjustments and a positive assessment, the required team size could be derived as the ratio between the estimated time duration and the desired (time-to-market) duration. The team size was then doubled, with the intent of catering for continuous pair programming. All the calculations were also done in a pessimistic view where duration was multiplied by 1.6, and in an optimistic view where duration was multiplied by 0.6—this determined lower and upper bounds on expectancies.

The PM Team continued working on prioritisation. As mentioned, each story was assigned the mean of the countries’ Benefit/Penalty rankings (relative to that story) that had been established autonomously the week before, and collected in the master spreadsheet. This created a base line prioritisation sequence. Now, the PM Team, collectively, expressed a desirability ranking of all stories, with the recommendation to change the base line Benefit/Penalty ranking as little as possible. The criteria used for expressing this desirability ranking related to each story’s market positioning, like: “*Differentiator*,” “*Competitive Response*,” “*Niche*,” “*Table Stake*,” and “*Spoiler*” (as initially hinted at in [11] and then described more extensively in

[5]). The main purpose of this step was to make the team appreciate the strategic reason (marketing positioning) why a story was present in the plan, and to allow to partition the entire story board according to such strategic views. This partitioning was going to be very useful later, when the PM Team was going to identify minimum marketable features. Again the participatory decision making techniques proved essential for rapid progress.

Defining the Economic Value of a Story The teams had identified over 200 stories, estimated their size in story points and expected implementation duration, expressed an overall prioritisation ranking, and had an initial team-size estimate to achieve the desired time-to-market. This result would be considered sufficient as an agile project plan. However, the task was to produce a business plan for investment appraisal.

To support the business plan, it was necessary to estimate the economic value of each story. The latest fiscal year's total revenue generated by all the original products was taken as a reference figure. An economic revenue value of a single story point was computed as the ratio between that total revenue and the total number of points.

$$\text{StoryPointValue} = \frac{\text{Revenue}}{\text{TotalNumberOfStoryPoints}}$$

It was assumed that the current market was willing to sustain the features described by all stories under consideration. The “worth” of a story was simply its number of story points multiplied by the story point revenue value.

$$\text{ValueOfStory} = \text{StoryPointValue} * \text{NumberOfPointsOfStory}$$

Re-Computing the Economic Value of a Story Point through Successive Refinement and Triaging Now was the time to discard stories that were worth less than others. This was a step wise refinement process. After a single story was eliminated, the economic story point revenue value was re-computed. In other words, the economic value of a story point changed continuously during this phase. Naturally, it increased because the reference revenue of the latest fiscal year was still the same, while the number of story points had decreased by those points assigned to the story that had been eliminated.

The PM Team could exercise an aggressive triage process: they realised that many stories were simply “*not worth their price*”: the market would not generate that revenue for that feature! In order to reach their conclusions, the PM Team often entertained additional discussions (typically via phone or instant messaging) with representatives from marketing, sales and accounting, both from the company's headquarters as well as from the single countries. Eventually a balance was reached between the economic value of the story point and all the stories left on the story board. At the end, the number of stories had been reduced to from over 200 down to 118.

Practical Tip: By using this method, you can reduce the scope and avoid feature creep. You can eliminate stories on the basis of an economic argument, rather than opposing opinions and power plays that would break down the team spirit.

Calculation with the Incremental Funding Method On the basis of the desirability ranking, the PM Team clustered all stories into sets of *Minimum Marketable Features* (MMFs) as described by [12] and [13]. Within each MMF, the stories retained the rankings that were attributed to them earlier. Once the number and scope of the MMFs stabilised, it was possible to apply the *Sequence Adjusted Net Present Value* (SANPV) calculations of the Incremental Funding Method (IFM) described in [12]. A final attempt at manually optimising the sequence was done. All these calculations were done with the involvement of a corporate accountant, who gave immediate feedback if they were not in line with corporate financial targets. After a few iterations, the order of implementation of the MMFs was established—even though it was not the theoretically optimal sequence, it was good enough to support a sustainable business plan.

4.2.5 *The Final: Project Approval*

During the last few days of the fourth week, the project plan was finalised and gained the unanimous consent of all product managers who expressed the recommendation to proceed with the operational project. At the same time, the business plan was finished with assistance of the company's headquarters office. Thereafter the two plans were sent to all country managing directors; who also considered and endorsed the plan from their viewpoint, after consulting with all local stakeholders. Shortly after, a Programme Board involving senior country representatives and senior officers from the company's headquarters met, and collectively expressed all involved countries' common and formal assent to the plans. The business plan was finally submitted to Executive Management who approved it primarily on the basis of its economic soundness. In addition to this, the fact that all participating countries unanimously sustained the proposal was seen as a major breakthrough. Likewise, the fact that the plan was produced in only five weeks, rather than several months of a roaming business analysts visiting all countries and elaborating project plans separately, was valued as great improvement over the previous attempts.

Having achieved such a positive outcome, it is interesting to compare and discuss the journey to implementing agility in an international setting and the benefits of agile approach with the traditional one. The comparison was a key argument when I presented the plan to the Programme Board and Executive Management.

4.3 Benefits from Implementing Agility over Traditional Approaches

The case discussed in this chapter is a real world case where an agile approach can be compared directly to at least one of the two earlier unsuccessful implementations of traditional approaches. Information regarding the first attempt is unfortunately unavailable; while the second one was made according to the PRINCE2 methodology, and in particular emphasising the *Product-based Planning* technique.

4.3.1 More Commonality

The PRINCE2 method enabled the project managers to define in detail the products to be delivered through a Product Breakdown Structure (PBS), with the intent of capturing all related work activities and intermediary deliverables. Each country had to employ the product-based planning technique. Subsequent analysis of the aggregated PBS determined the degree of commonality between all country requirements.

The analysis revealed that approximately 30% of all requirements were common. Most of the common requirements were non-functional requirements (relating to infrastructure, rather than to the problem domain). By examining the respective PBSs, the new products to be developed for each country were deemed to be broadly in the same order of magnitude of scope. In other words, the amount of work required to deliver each single product was approximately the same. Therefore, given 100 as the size of any such product, a broad estimate of savings for eight countries was computed as:

$$\frac{100 \times 8 - ((100 - 30) \times 8 + 30)}{100 \times 8} \times 100 = 26.25$$

The common project would require 26.25% less effort than eight distinct country level green field developments. The savings was naturally significant. However, since the idea was also to reduce total head count and utilise one team (rather than eight distinct teams), the overall time to market was deemed too high—the team could focus on one country at a time only. The total time estimate diluted the break-even/ROI calculations beyond acceptability. Even worse: some countries at the end of the implementation queue would have been forced to wait beyond the end of life-cycle of their current products, exposing them to the risk of being uncovered on their markets! Naturally, the project was not approved.

With the agile approach, the amount of commonality between the countries—which was derived directly from the stories being tagged as of interest by all countries—was much higher. Over 80% of all stories were recognised as of interest by all countries. Another 5% ca. were considered of interest at least by two or more countries (but not all). The remaining 15% ca. were truly single country specific. By applying the original savings formula, the savings was determined as:

$$\frac{100 \times 8 - ((100 - 80) \times 8 + 80)}{100 \times 8} \times 100 = 70$$

Therefore, 70% would be saved on the hypothesised distinct country level green field developments; this compared very favourably with the 26.25% savings of the traditional plan.

4.3.2 Smaller Scope

One reason why the degree of commonality was much higher with the agile approach is the reduced scope of the project. The story cards had been reduced from over 200 to 118. Assuming those initial 200 stories represent the total requirements that had been analysed with the traditional approach, the decrease in scope is approximately 40%. The total estimated effort was much smaller than what had been estimated by the traditional approach, with an immediate impact on resource requirements and time-to-market.

4.3.3 ROI Anticipation

The usage of MMFs was important for the project's economic viability. The project was partitioned in 11 MMFs, of which the first contained "core" functionality. The second MMF was deemed as sellable: as soon as it was implemented, revenue could be generated. This would allow ROI to be anticipated, without having to implement the entirety of the project—as was required by the traditional plan. Furthermore, by showing the SANPV of the MMFs, it could be sustained that a (near) optimal revenue generation plan had been derived; at least it was demonstrably better than any other alternative.

4.3.4 Smaller Country-Specific Dependencies

Only 15% of stories were truly country specific (on average this is less than 2.5 stories per country). This would allow for small "localisation" teams (even a single developer) to be set up for each country, and quickly supplement the main team's work with localisation code. On the one side, this would reduce head count drastically in most countries, with corresponding cost savings. On the other, this would enable parallel development of all country versions. Most important, no country was at risk of being the "last in line" and loose market opportunities because of unattainable development timelines.

4.3.5 Avoiding Waste Upfront

The attribution of an economic value to the story could be done—quite obviously—only after it had been estimated. Although time was spent on the estimation activity

as part of the planning game, by attributing an economic value to each story, it was possible to decide which stories were worth keeping, and which ones could be discarded. The crucial point is that it was not necessary to have to code and implement the stories only to discover—later—which ones would have been wasted effort.

4.4 Why Agile Succeeded?

The salient question is how come the results of the traditional and the agile approach were so dramatically different, while the underlying *problems* and *people* were all the same? One well known statement of the agile Manifesto in [14] is to value “*Individuals and interactions over processes and tools*”—and this is a point where the effects of such a viewpoint made a difference. The following success factors stand out:

1. Induction to learn about participatory decision making;
2. Co-location and alternating on/off-site activities;
3. Promiscuous pair story authoring;
4. Economic value of Story Points.

While the first two were simply the consequence of experience and insight, the last two were innovative. Let’s examine all these factors in turn.

4.4.1 Induction

The purpose of the three day workshop was to introduce agile concepts to the team members. The subtler and more valuable activities were about introducing the team to participatory decision making techniques. The insight gained about participatory decision making techniques helped the teams to proceed very quickly during the weeks when they were required to work together. Most strikingly, there was not a single instance where the flow of work got stuck because of people arguing in defensive ways about their own positions and points of views. This had been a problem in the earlier attempts, where a common agreement was never reached. Often, in the earlier attempts, the larger countries wanted to have a lead role, and thus created conflicts amongst themselves. Likewise the teams did not disperse energies in any *bike shed arguments*. Instead, all differences in opinions were seen as obstacles to be overcome as quickly as possible; or as opportunities to gain deeper insights into other’s viewpoints. All energy was spend effectively, rather than wasted in unproductive disputes.

The importance of this initial workshop cannot be stressed enough. It was a hard call to involve all these people with different backgrounds, experiences, cultures, etc. and make them work together as a team after only three days, with an approach

(the agile one) that none had worked with before. There was a huge risk that the agile approach could have been perceived as “enforced” upon them, and hence rejected.

The workshop focused on two major topics: the nature of software and participatory decision making. By making the team members understand more about the nature of software, and how agile processes actually work *with* rather than *against* that nature, they were prepared—intellectually—to accept the new ways of working. By making the team learn to perform participatory decision making, they could overcome the barriers that separated them one from the other. The exercises of the workshop made the team appreciate and want—emotionally—to try the agile approach. The agile approach was not perceived as enforced, but effectively as chosen by the team members themselves, because they gained the understanding and insight about why this was beneficial for them.

The most valuable outcome of the workshop was to produce the “buy-in” from all of the team members; and ability to reflect this positive attitude back home. From a distributed, multi-site perspective, this in turn, made the managers in the home office more interested in seeing how the project would actually evolve, laying the groundwork for the feedback loops that eventually took place between the teams and their home offices.

Practical Tip: When assembling new teams, primitive team “building” exercises are performed to make the team “gel.” Unfortunately, most of such exercises have very little effect on how the teams will perform later, when they will have to do their real work. It is much more effective—as in the studied case—to teach the team about what problems they will encounter when working together, and how to recognise and resolve those problems once they arise. It is important to involve the team both on an intellectual (“understanding”) level, and on an emotional (“wanting”) level so that they are prepared to do the work.

4.4.2 Co-location and Alternating On- and Off-Site Activities

Getting all participants together to meet and interact with each others was one major difference with respect to the earlier attempts. The communication patterns that are typical of an agile setting would not have been possible without the frequent face-to-face interactions. The scheme of working by alternating weeks in the home offices and weeks in off-site co-location, where all team members got together, was extremely important. The most significant effect was seen in the “home” offices, where it was possible to gain the buy-in of all stakeholders, and keep them involved in the whole planning process.

The situation was very different from the earlier traditional approach, when a roaming business analyst had been visiting all countries, and interviewed the product managers alone as a means to elicit and collect requirements. Typically, the

countries could express their viewpoint only once, and then were practically excluded from any further involvement and interaction. This naturally had the effect of inflating all requirements, since everybody was concerned about getting everything they could think of into the overall plan. Also, while only the product managers were interviewed to obtain the requirements, all other stakeholders were not involved: it was simply assumed that the product managers were effectively representing them.

In the agile approach, the product managers and software engineers who were sent to the workshops acted as “ambassadors” of their country offices. The alternating between on- and off-site work realised feedback loops with the stakeholders at home. The stakeholders were much more involved, informed and could effectively interact with the work in progress. As a result, the plan produced by the team was much more likely to be well accepted back home.

It is interesting to consider if this kind of result could be achieved without co-location. Given the collaboration tools that are available today, it is fair to assume that pair authoring can be technically implemented without co-location. However, after having seen how the team performed, I am convinced that you can achieve the productivity and unanimity only when people are co-located.

Practical Tip: Go beyond travelling managers; involve other staff too! In this case, product managers, project managers, domain experts, accountants, senior and junior software engineers were sent to the off-site location. Furthermore, there were frequent and intense interactions with other stakeholders at “home,” such as the country and business unit managers, and staff from sales, marketing, and support.

4.4.3 *Promiscuous Pair Story Authoring*

The major breakthrough in this success story is the practice of promiscuous pair story authoring. The tangible impact was in determining the amount of commonality between the countries; the subtler aspect was *how* that commonality was established.

In the traditional approach commonality was identified through analysis of requirements that had been elicited by a business analyst. The traditional approach tried to *extract* commonality from the elicited requirements. The requirements were expressed on the basis of *past experiences* of each product manager, with a perspective that was restricted only to her/his own country.

In the agile approach, the effect of promiscuous pair story authoring was that all product managers effectively *created* a common and shared vision of the future product. This is the key point: commonality was not sought after or extracted. Instead, it was *created* by the very process employed, which fostered a collective *future vision*, with *shared understanding* and *intent*.

There were many times, during the promiscuous pair story authoring sessions, when team members went through “Aha!” moments, as they discovered that similar

concepts and ideas were described by different words in the various countries, and were even framed in totally different contexts. The overcoming of such apparent differences contributed a great deal in creating a common and agreed upon domain vocabulary, as advocated by [15]. The team members were able to conceive of entirely new ideas. At other times they merged concepts and earlier requirements that they had brought from their own experience. The resulting *shared creation* nurtured sentiments of enthusiasm, ownership and buy-in by all participants. They quickly felt that they all, together, really owned the project. This sentiment of shared experience would never have been possible without the aggressive promiscuous pair story authoring, with pairings changing as often as every 15–20 minutes. It also occurred that some product managers realised that other colleagues from other countries had so much better ideas than their own, in some particular requirements area, that they voluntarily scrapped their own proposals and fully adopted their colleagues' ones. This kind of realisation would have been entirely impossible with the traditional approach. It is explained by the rapid knowledge transfer enabled by exploiting the beginner's mindset when forcing the least knowledgeable member of the pair to write the stories.

4.4.4 Economic Value of Story Points

A crucial stratagem was the effort of making business value explicit and measurable by attributing an economic value to the Story Points. The idea was suggested by the Theory of Constraint's focus on throughput as described by [16]. For User Story based software projects, throughput was defined by [11] "*as the value of the delivered Stories. The value is the sales price (or budget) less any direct costs [...]*"

For the purpose of triaging the stories, it was sufficient to consider the *market value* of Story Points, and not necessarily throughput. The essence was that a *cost* figure was not assigned to Story Points. Instead the *market value* of Story Points was defined relative to the latest revenue figure. Once the market value of the stories was established, it was possible to recognise the impact of triaging, and *this* allowed to reduce the overall scope.

4.4.4.1 Avoiding Feature Creep

Notice that this is very different from other established agile approaches too. They define a cut-off point in the backlog of stories, identified by velocity measurements, to decide what to include and what to exclude from a particular release. In this case an effort was made to explicitly eliminate stories, effectively reducing the overall backlog and the entire scope of the project, and avoiding *feature creep*. In the traditional approach, triaging would have been perceived as an attempt to penalise the "owner" of the feature being eliminated. It would have provoked defensive reactions—with the negative consequence of triaging not being exercised at all.

4.4.4.2 Increasing the Value of a Story Point

In an agile setting, but with a conventional cost-based approach, the production of a story point would always cost the same and therefore would be unaffected by triaging. The only way to get “more value” would be to achieve a lower cost per story point, or increase velocity/productivity in terms of work delivered per unit of time. This is quite different than increasing throughput in terms of real economic value of the story point, as in this case. Most of the stories that were eliminated by triaging were those that were worth less (relatively to their ranking and degree of commonality, of course). Often these were the stories that had been proposed by one (or a few) of the participating countries. However, since the rationale for eliminating the story was based on economic value, this did not provoke defensive reactions; rather the outcome was well accepted, because the stories remaining after the triage were characterised by higher and higher economic value.

4.4.4.3 Virtuous Circle

The elimination of stories automatically increased the return on investment. The reason can be found in [11]. The ROI of a software release, from the throughput accounting perspective is defined as:

$$ROI = \frac{\textit{Throughput-Operating Expense}}{\textit{Investment}}$$

This last point is worth stressing: eliminating stories has a direct impact, from a TOC perspective, by reducing the Operating Expenses (i.e. the effort needed to produce the software). This obviously increases the numerator in the ROI ratio; and hence it is a practical and financially grounded way to “*maximising the amount of work not done*” as it is formulated in one of the twelve principles of the Agile Manifesto by [14] and effectively applying the Lean principle of eliminating waste. In practice, by continuously re-computing the value of a story point, it became *visible* how eliminating stories increased the overall value of the entire project.

Finally, by actively reducing the amount of work by eliminating stories, the implementation time and consequently the time to market would be reduced proportionally (naturally, assuming constant velocity). Therefore (in addition to the already mentioned effect of ROI anticipation due to the use of MMFs) ROI would not only increase, but it would also be reaped much earlier.

4.5 Conclusions

While the derivation of the final plan went through a lot of work, with many persons contributing, the key phase was during the practice of promiscuous pair story authoring. It was during that stage that the team really “gelled” and came together. While the induction established the collaborative and participatory frame of mind

for all subsequent activities, it was the promiscuous pair story authoring that truly changed the team members' attitude: it changed from scepticism to a convinced "can-do" attitude.

The main challenge was persuading the product managers of each product and country about the opportunity, necessity and viability of developing a single platform for all of them. In fact, at the outset there was strong scepticism about the possibility of finding common grounds amongst all countries. Through the promiscuous pair story authoring practice, all scepticism was overcome by the team collaboration, and by the team succeeding in creating a shared and common vision. That vision then inspired the rest of the activities.

Assigning an economic value to story points allowed the teams to reduce the overall scope, avoid feature creep, and do so according to sound economic criteria, rather than opinions, hence avoiding many conflicts that would have appeared otherwise.

The case shows how practices that matured in the field of coding (collective ownership, pair programming) can successfully be adapted and applied in other phases of the life-cycle (requirements gathering). The case also illustrates how some agile principles, and in particular paying attention to the people involved and teaching them to collaborate, and then actually giving them the opportunity to put those notions into practice, can radically change not only how the project plan is materially made, but also the final and financial outcome.

Acknowledgements The author wishes to thank Dr. Bruce Sharlau, Teaching Fellow in the Computing Science Department at the University of Aberdeen, for reviewing and commenting on the manuscript.

References

1. McGrath, R. G., & MacMillan, I. (1995). Discovery driven planning. *Harvard Business Review*, July–August 1995.
2. Austin, R., & Devin, L. (2003). *Artful making, what managers need to know about how artists work*. Upper Saddle River: Financial Times Prentice Hal.
3. Cohn, M. (2004). *User stories applied: For agile software development*. Reading: Addison-Wesley.
4. Kaner, S., Lind, L., Toldi, C., Fisk, S., & Berger, D. (1998). *Facilitator's guide to participatory decision-making*. Gabriola Island: New Society Publishers.
5. Cohn, M. (2005). *Agile estimating and planning*. New York: Prentice Hall.
6. Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading: Addison-Wesley.
7. Belshee, A. (2005). Promiscuous pairing and beginner's mind: Embrace inexperience. In *Agile conference 2005*.
8. Belshee, A. Promiscuous pairing and the least qualified implementer (podcast).
9. Wieger, K. E. (1999). First things first: Prioritizing requirements. *Software Development*, September 1999.
10. Wieger, K. E. (2003). *Software requirements* (2nd ed.). Redmond: Microsoft Press.
11. Anderson, D. J. (2003). *Agile management for software engineering: Applying the theory of constraints for business results*. Englewood Cliffs: Prentice Hall PTR.

12. Denne, M., & Cleland-Huang, J. (2003). *Software by numbers: Low-risk, high return development*. New York: Prentice Hall.
13. Denne, M., & Cleland-Huang, J. (2004). The incremental funding method: Data-driven software development. *IEEE Software*, 21, 39–47.
14. Beck, K. et al. (2001). *The agile manifesto*.
15. Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software*. Reading: Addison-Wesley Professional.
16. Goldratt, E., & Cox, J. (1992). *The goal: A process of ongoing improvement* (2nd revised ed.). Great Barrington: North River Press.

Chapter 5

Scrum and Global Delivery: Pitfalls and Lessons Learned

Cristiano Sadun

Abstract Two trends are becoming widespread in software development work—agile development processes and global delivery, both promising sizable benefits in productivity, capacity and so on. Combining the two is a highly attractive possibility, even more so in fast-paced and constrained commercial software engineering projects. However, a degree of conflict exists between the assumptions underlying the two ideas, leading to pitfalls and challenges in agile/distributed projects which are new, both with respect to traditional development and agile or distributed efforts adopted separately. Succeeding in commercial agile/distributed projects implies recognizing these new challenges, proactively planning for them, and actively put in place solutions and methods to overcome them. This chapter illustrates some of the typical challenges that were met during real-world commercial projects, and how they were solved.

5.1 Introduction

Two trends are becoming widespread in software development work—agile development processes and global delivery (i.e. development teams who are spread over geographically wide areas, typically crossing country borders).

The former—especially the Scrum process [1]—promises an increase in productivity, quality and value of the software developed; the latter promises both sizable cost reductions and a mitigation of skilled personnel shortage issues in certain countries or markets. Combining the two is therefore a highly attractive possibility.

However, certain conflicts exist between the assumptions underlying the two ideas. Scrum consists of a set of organizational techniques and collaboration mechanisms mainly target to local teams working in close physical proximity. Such techniques and mechanisms assume that, in most circumstances, an empirical,

C. Sadun (✉)
Tieto Norway AS, Oslo, Norway
e-mail: cristianosadun@hotmail.com

communication-heavy and feedback-based process is better suited than a defined-requirements one to develop nontrivial software. Offshore software development is often executed with a “detailed requirements” approach, possibly to minimize the risks due to distance and higher communication barriers (from operational ones—such as different time zones or different levels of proficiency in the team language— to cultural and attitude differences).

Furthermore, when global delivery is used in realizing a piece of software, these issues can be compounded by other factors: expectations about global (offshore) delivery centers as production factories; different levels of understandings and expectations of the level of agility of the process among different sub teams; the stronger emphasis typically placed on defined communication (and related artifacts, like project reports, meeting minutes, detailed changes documentation), in the attempt of achieving better control (or assuage fears of failure) over far-away, independent development groups.

While early research and guidelines exist on how to distribute Scrum-based software developments (typically via Scrum-of-Scrums [2]), when Scrum and global delivery are mixed, there are still risks of a gap of expectations and assumptions between various subgroups of the development team when agile processes are combined with global delivery.

Based on the experience of some custom-developed application projects delivered in 2007/2008, in this chapter we will explore some challenges and pitfalls which we met during the concrete execution of this kind of work, their consequences and which actions mitigated them and what was the final outcome.

5.2 Cases Overview

5.2.1 Background

This chapter is based on two fixed-price projects, which we’ll call conventionally NOR1 and NOR2. The projects were delivered by a professional services company (“Supplier”) to two different organizations (“Customers”) in Norway. The tables below outline the size and attributes the Supplier organization.

Table 5.1 Case synopsis

Company: NORSupplier ^a	
Number of developers	10.000+ (international)
When was agile introduced	since 2005, but in separate parts of the organization
Domain	IT consultancy and services

^aThe name is changed due to confidentiality reasons

The customer organizations (“NORCustomer1” and “NORCustomer 2”) were in both cases the Norwegian arm of a large multinational (about 25.000 + employees for “NORCustomer1” and 42.000 for “NORCustomer2”).

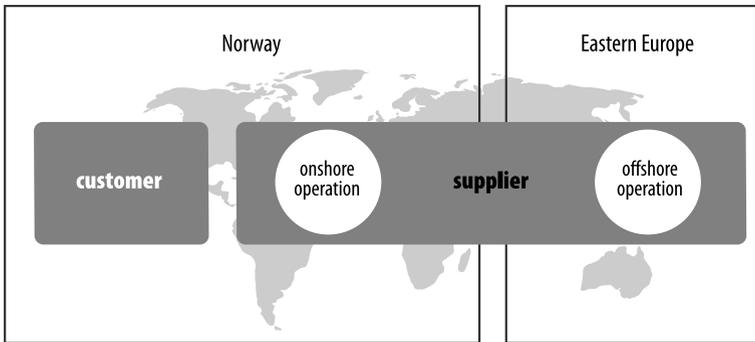


Fig. 5.1 Project partners

The goals of the projects were nontrivial, but well within the complexity and size of typical modern software engineering efforts. Both Customers were unused to agile methods, having applied them only in pilot or test form in their internal IT departments, and certainly not within the Customer/Supplier context, typical of professional services, where deliverable software is expected from a Supplier at a target price within a target date and with a defined quality level.

The deliveries implied distribution of work between two onshore and offshore locations, and in both cases, the Supplier’s Norwegian operation was designated as the “onshore” one, and the other Supplier’s locations where “offshore”—i.e. remote delivery units. The figure below illustrates the setup.

Both projects were ultimately successful—both for the Customers (which obtained the deliverables they had purchased) and for the Supplier (which made a profit and obtained valuable references), and—in agile spirit—for the participants, which (according to a survey taken at the end of both project) felt the projects rewarded and benefited them in terms of individual development and experience. However, this success was far from assured during the projects execution, and active steering was necessary to achieve the final result. This chapter deals with some of the challenges we met, and the solutions which ultimately proved successful.

The tables below summarize the two projects.

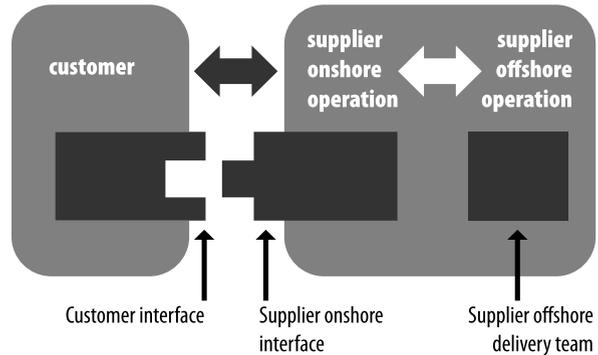
Table 5.2 Project 1: overview

Project NOR1	
Duration:	12 months
Status:	finished
Agile practices:	Scrum (full implementation)
Involved locations:	Norway, Eastern European country

Table 5.3 Project 2: overview

Project NOR2	
Duration:	four months (first phase)
Status:	finished
Agile practices:	Daily meetings, user stories-like requirements, short iterations, planning gatherings
Involved locations:	Norway, Baltic country

Fig. 5.2 Work distribution



5.2.2 Project NOR1

The first project (Project NOR1) was delivered to NORCustomer1 and aimed to replace an existing sales web portal with one having noticeably improved usability, while also consolidate functions from other web portals (dealership information, campaigns, product information) in one single tool. The user base for the portal consisted of sales people in the customer’s dealership organization; they can both use the portal functionality to close/register sales in realtime, with the customer in attendance, and gather back office information such as sales reports, commission reports and so on. The Customer was a middle-size organization (about 700 people) fully owned by a much larger Nordic company.

From a technical point of view, the project primarily included:

- the development of the portal application;
- the replacement of an obsolete sales automation process with a novel one—and the consequent phase-out of the integration platform supporting it, in favor of a newer SOA platform (already well established and mostly developed);
- integration of CMS (Content Management System) functionality for supporting information channel;
- development/integration of administration functions.

Scrum was selected as development methodology, primarily due to the uncertainties about the final scope, the prioritization of the consolidated features and

the technical challenges in the integration platform, all of which could not be predicted reliably at project start. A typical example was a set of features concentrated over Christmas-time campaigns, whose realization complexity was difficult to assess with an adequate degree of confidence by Customers personnel, and for which the tradeoff between features and cost could not be readily identified at start. In this project, all the practices prescribed by Scrum were adopted.

Project NOR1 duration was of about one calendar year. The project was heavily distributed at the Supplier side, with a 24 to 20 people staff, and thus a work distribution over two locations; in the Supplier organization, the ratio of offshore to onshore personnel (in terms of headcount) is called offshore ratio. Project NOR1 had with an initial offshore ratio of approx. 80%. I.e. 4 people onshore, 20 offshore, initially. Note that the number (and thus the ratio) changed slightly over time, to a 2 onshore/18 offshore at the last implementation Sprint.

The chosen Supplier's offshore site was in an Eastern European country in the same time zone as Norway. The site was chosen primarily for the following reasons (in order of importance):

- location maturity *The site was the first offshore center established by the Supplier and the one with the largest history of operation;*
- available competence types;
- ease of communication due to the time zone.

The offshore staff was divided in three Scrum teams, including junior and senior developers. The onshore roles included initially a functional/technical architect, a project manager, a co-project manager in charge of accounting and budgeting, a test manager, and Scrum masters. The roles were distributed over four people, where three took the Scrum master role in addition to other as of the following table.

Table 5.4 Project NOR1 team

Location	Number of members	Roles
onshore	2–4 ^a	project manager, scrum master, technical architect, test manager
offshore	18–20	coordinator, developers, testers

^aThe number and composition of the teams were modified during the project

Such organization was set up as a tradeoff between an ideal organization (one person-one role) and the type, number and availability of personnel at project start; the fact that both onshore and offshore line management were stakeholders in the project execution; the internal contractual agreements between onshore and offshore subunits of the Supplier; and, last but not least, strong views on project control and accountability held by the onshore unit top management.

It is worth noticing that these are typical real-world constraints that may preclude theoretical ideal organizational setups, even in cases where theoretical or research guidance exists.

5.2.3 *Project NOR2*

The second project (Project NOR2), delivered by the same Supplier to NORCustomer2, aimed to realize an order management system for a new range of products in the Customer organization, automating order acquisition, validation, processing and fulfillment. No GUI component was to be delivered for the core system, but ancillary monitoring, control and recovery tools and their GUIs for ensuring accurate processing were also required and delivered. The Customer organization was a large company in Norway (with over 6000 employees in the country and several more internationally).

Technically, the project intended to deliver:

- a novel platform for order validation and processing at the Customer, based on explicit workflow management via a BPEL (Business Process Execution Language) engine as opposite to traditional process hard coding;
- integration with existing back-end systems as necessary to fulfill orders;
- a monitoring and recovering user interface for blocked or incorrect orders.

The project was intended also as a proof of concept for the platform, to validate its viability as a replacement for the more traditional, mainframe-based one.

Project NOR2 was actually a subproject in a much larger endeavor at the Customer organization. As it was on the critical path for a number of other sub-projects, there was a particular emphasis to delivery time and quality of the result, i.e., other subprojects depended heavily from it timely completion in order to start or to complete.

Similarly to Project NOR1, for Project NOR2 there was a receiving organization at the Customer. Due to the nature of internal agreements in the Supplier organization, only a subset of Scrum elements was selected, specifically:

- daily Scrum meetings;
- a work breakdown structure resembling user stories, that is—splitting functionality in roughly estimable, conceptually self-contained pieces. Test criteria specifications, however, were not explicitly added to each piece;
- partial production level deliveries in short iterations (4/5 weeks).

The rationale for selecting agile elements was that time scheduling was very tight, and strong progress control was deemed to be a major mitigation of delivery delays issues. However, due to contract expectations, a full Scrum could not be selected. In opposition to Project NOR1, a standardized contract was used, which contained a legally binding declaration of scope of delivery in terms of a fixed set of requirements to be realized, with no choice for dropping, replacing or adding new during the project execution. The reason for including Project NOR2 here is that there is empirical evidence of many projects taking this partial approach—so the relative findings can be interesting.

The duration of Project NOR2 was of about four calendar months. The project had a much smaller number of participants (2 onshore and 5 to 4 offshore) than Project NOR1, with an offshore rate of about 60%.

The offshore site was located in another country in Eastern Europe—this time, in the GMT+2 time zone. In Project NOR2’s case, the chief reasons for selecting the site were:

- concentration and availability at the site of high-level competences for the chosen technical platform (*the site is the Supplier’s “centre of excellence” for the type of technology which constituted the base of the solution*);
- ease of communication due to the time zone (still quite comfortable, with just one hour of difference from Norway)
- the good reputation for successful deliveries of the site, even if it had been existing for shorter time than the one selected for Project NOR1.

In Project NOR2, one single team took care of the delivery, with the offshore team consisting (as for Project NOR1) of senior and junior developers. The onshore roles were as of the following table.

Table 5.5 Project NOR2 team

Location	Number of members	Roles
onshore	2	project manager, technical architect
offshore	5–6	coordinator, developers, testers

5.3 The Experiences

5.3.1 Signing Agreements

Internal agreements between units were found to have a large effect on the construction and efficiency of the project organization.

5.3.1.1 Challenges

By definition, distributed work involves at least two separate country sub-organizations, which in order to collaborate need to undergo relatively precise contractual agreements. Two kinds of agreements were available when both Project NOR1 and NOR2 started: *a resource-level agreement* and *a project delivery agreement*. With the former, individual people are hired in by the onshore unit and the relative hourly costs invoiced to it; with the latter, the offshore unit retains direct control of the offshore people but takes explicit responsibility in delivering desired artifacts and coordinating with onshore. For Project NOR1 onshore and offshore entered a resource-level agreement, and a project-delivery agreement for Project NOR2.

Resource-level agreements allow tighter formal control of the project and comply with the Scrum precept that developers and other assets should be allocated 100% to the project. However, we found that there was less incentive for offshore management to provide the best people and to particularly stimulate them to participate with the intensity that Scrum requires—since the agreement does not include direct responsibility for the final delivery. The responsibility for ensuring (initially) the suitability of resources, and (during the project) their motivation and coordination fell mostly on the onshore organization.

Project delivery agreements hold greater incentives in the offshore organization to take direct responsibility for the outcomes, but staff allocation was handled by the offshore units, so that it was not entirely dedicated to the project but often split between several ones. In Project NOR2, we experienced initially lags in communication, missed participation to daily Scrums, unavailability of people in planning poker sessions and the likes, which could be traced back to this kind of agreement. The major consequence of this was in increased load in the onshore Scrum master/Project manager, who had to spend considerable energy in “hunting” people, and the initial production rate, which was lower than expected. While a team velocity is usually *observed* more than *estimated*, in Project NOR2 case both time reporting and random interviews made clear that the production was not happening at full capacity due to the fact that people were not dedicated solely to the project.

5.3.1.2 Solutions

Formal agreements, even if internal, are not generally easy to change midway, so no specific action could be reasonably taken to address the problem specifically; however, we resolved to interpret both situations as a result of both formal agreements and personal motivation of individuals on the onshore and offshore teams (as opposite of only the formal agreements).

We thus had a twofold approach: on one side, we made active use of both available contractual tools and internal negotiations to remove/reduce personnel who was formally allocated to the project but in practical terms was not able to perform adequately; on the other, we resolved to address explicitly motivation and commitment, attempting to establish the feel of a localized Scrum team by traveling often on location in both directions, and explicitly emphasize (both in words and actions) the peer-to-peer relationship between onshore and offshore—all aspects which are addressed in the following chapters.

Both activities resulted in overhead costs (at both business and project management levels and for additional traveling) but succeeded in correcting the situations.

5.3.1.3 Lessons Learned

As a lesson for future projects, mixed resource/delivery and/or shared-risk internal agreements are being examined by the Supplier, so to provide the opportune incentives to both onshore/offshore organizations for the mix of delivery responsibility

and resource allocation that is required from Scrum-based efforts also in distributed situations.

A practical tip to address these issues is perhaps obvious: ensure to establish good personal relationship with the offshore site management, because in some form you probably are going to need it.

5.3.2 Establishing Remote Access

Both projects required work either at Customer's premises or in connection with Customer's systems. For non-distributed Scrum effort, this is normally a non-issue: a suitable location at the Customer's premises is found, where the entire team or teams can work. Often, offshore work is instead based on the idea of shipping specifications, perhaps organizing more or less frequent clarification meetings and receiving software or other deliverables.

5.3.2.1 Challenges

In both projects, this was simply not possible: agility required that offshore developers have the same view of customers system as people onshore, share the same code repository, access logs of daily builds and test runs and so forth. This implied setting up some form of remote access. Technically, this was relatively feasible, but we encountered two major challenges:

- for project NOR1, security policies on both Supplier and Customers were not suited for this kind of remote access, and special policies had to be drafted and approved to allow controlled exceptions to such policies;
- for project NOR2, the processes in the Customer's operations unit in charge of the necessary physical configurations (firewall openings, virtual network clients setup, and verification of compliance of Supplier's workstation to acceptable parameters) were not suited to the tempo, speed and productivity expectations of agile efforts—and thus were lagging noticeably in time at the start of the project. In practice, bureaucracy risked to choke the project progress.

Both aspects resulted in delays at project start, which put at risk the productivity claims made for agile methods and setups.

5.3.2.2 Solutions

The first challenge was met by drafting temporary agreements which allowed existing security policies bypassed under direct responsibility of high level officers at both Customer and Supplier. The actual final agreements did not become ready until the projects were well under way.

The second challenge was addressed, in a somewhat less systematic way, by using the Supplier's contact network at the Customer to speed up things.

For project NOR1, the overall time to establish the necessary communication ended up to be about 5–15 calendar days (not all the systems became accessible at the same time), with an overall delay of about a week; for project NOR2, making use of the contact network reduced the time necessary for the necessary network adjustments to a couple of hours. However, the total delay was also of about a week, since that time had been spent in waiting for the formal Customer process to take place before actually attempting to contact directly the relevant Customer personnel.

5.3.2.3 Lessons Learned

The most obvious lesson is to assume that support and operations functions at a Customer, especially one not used to work in agile ways, will have a response time tailored to more traditional software development cycles. Therefore, requests such as firewall openings, hardware ordering, router configurations and so forth need to be issued well in advance of the first iteration/sprint, otherwise productivity will suffer. A strong contact network at the operational level at a Customer can, of course, be extremely helpful, as it happened for Project NOR2.

Regarding the legal agreements, there is hardly a way to speed them up if they are not already in place when the project is being negotiated between Supplier and Customers; however, generally project owners or Scrum Customer Representatives in the Customer organization have an interest in the project progress as strong as the organization that is delivering it. Therefore, they can generally become its allies in finding interim solutions and/or work within their own organization so that a viable solution is found. The tip here is simply to raise the issue as soon as possible with a customer stakeholder who has both an interest in the project success and the authority to push or approve interim solutions.

It's interesting to note that these issues were much easier for subsequent projects with the same Customers, but for future projects with new customers, the calendar planning (as opposite to the effort estimates) will have to take such delays into account.

5.3.3 *Overcoming Communication Barriers*

Scrum implies tight communication within a fully dedicated project team, both formal (e.g. daily Scrums, planning meetings etc.) and informal. When executing it with people distributed over different geographical locations, it quickly emerged that maintaining informal communication was difficult.

5.3.3.1 Challenges

Both projects started with a short visit of selected customer representatives to the main offsite locations quickly followed by full-team onsite kick-offs (in Norway) to allow personnel to get to know each other (and Customer's representatives) and develop a sense of teamwork and mutual commitment. However, even if both onshore and offshore personnel were on average enthusiastic and committed about the adoption of agile processes, it soon surfaced they had slightly different expectations and assumptions on how in practice to execute it; and language and culture barriers had a bigger impact than initially thought.

Repeated attempts to supply technical tools to facilitate communication (from chat systems to web cameras to issue tracking systems customized for the project) did not seem to produce measurable effects. Perhaps predictably, exhortations to increase the level of communication did not have any better effect. Communication simply did not seem to happen by itself—even when tools were available, people were not using them to the degree that seemed desirable and necessary.

5.3.3.2 Solutions

Three main measures were taken to compensate for this and did have the desired effect of explicitly *drive* communication:

- on the offshore side, three senior developers were given the roles of local coordinators or co-Scrum masters, taking a similar role of “obstacle remover” in the physical offsite environment.
- The amount of physical traveling was also increased—with more frequent travels between key personnel offshore/onshore and vice versa than originally estimated. Both projects had a travel plan. However, Project NOR1 had explicitly planned and priced a set of periodic travels between offshore and onshore throughout the entire effort, making this a transparent cost-driver for the Customer, while for Project NOR2 the Project manager had simply budgeted a cost as a percentage of the overall project value in order to determine the overall project cost and allocated most of it to the initial kick-off.

On hindsight, the first approach allowed more flexibility than the second: firstly, since communication implied representative of both Customer and Supplier, they both had a joint interest taking the appropriate steps and allow altering of the travel plan; secondly, the tradeoff between cost and amount of travel was easier to handle, since with the second approach any rescheduling of travel represented a cost incurred solely by the Supplier.

- In Project NOR1, about two months into the project calendar schedule, the onsite architect was replaced by an offshore person, who however moved onsite for the duration of the project. The intention was to allow someone from offshore participate directly to the initial period of each sprint (with the accompanying clarifications and informal communication involved), so to ease communication to offshore bypassing language and culture barriers.

It is important to note that this person was chosen carefully for his communication abilities and the degree of respect he commanded among the developers in the offshore unit.

A reverse approach was taken in Project NOR2, where the onsite people spent noticeably more time at the offshore operation, while a local coordinator (who had been ineffective due to overcommitment to other projects) role was replaced by direct coordination by the onsite Project manager—about two months into the project. Since the delivery group was smaller, the onsite personnel had already established good communication channels to offshore, so in this case the intent was to provide the physical opportunity for informal communication to happen.

Both approaches proved successful—the degree of communication increased to a more than workable degree by physically exchanging a small amount of people from offshore to onshore or vice versa. Since the time span of the two projects were appreciatively different, it is possible that one of the two approaches is better than the other, but no evidence of this can be extracted from the two cases.

5.3.3.3 Lessons Learned

Communication and culture issues are best addressed explicitly at project start, but this is not sufficient. A few selected persons at both onshore and offshore location need to be explicitly trained and followed up on facilitating communication. For them, planned, periodic travel with enough frequency (for example every four or five weeks) seems to be a better idea than an aggregated, general traveling budget which usage is determined during the project. Moving a representative of the off-site location onsite or vice versa seems for the project duration is an excellent, if expensive, mechanism for facilitating communication, so long he/she his chosen for his/her communication and bridge-building skills in addition for other technical or management competences. Such bridging is much easier if the person has informal authority between his/her original location. Our experience suggests that one person is enough for groups of up to 20+ developers, at least so long the project execution time gives him/her the opportunity to get to know the “other site” counterparts well enough.

5.3.4 Actively Managing Distributed Agile Projects

Agile approaches in general are relatively new, and agile distributed approaches, especially in a Supplier/Customer context, are even less common and documented. As a consequence, there is less established guidance on the definition and interconnection of individual project roles or the best junior/senior competences mix at offshore sites, and we found that the personal attitudes or skills of project personnel was a key issue to achieve the expected productivity and results.

5.3.4.1 Challenges

In Project NOR1, we found that the initial junior/senior mix was too skewed towards junior competences. Simply put, many senior developers were spending too much time coaching and correcting work of junior people. While this is not a challenge of agile or distributed projects per se, the distribution aspect compounded it: the lack of physical co-location and direct observation (and perhaps a natural tendency of team members of supporting each other) delayed the recognition of the problem.

5.3.4.2 Solutions

During the project, corrective action was taken by asking the Scrum teams to identify which people were contributing less, and as a consequence four people were reassigned to other internal projects, where they had a better opportunity to gain experience. Productivity (measured in story points) did rise of about 30% on average on the remaining sprints (rising from about 8 to 12 story points per week). While of course this was not an experiment in isolation and other causes may have brought about (or contributed to) the productivity increase, there were clear signals and evidence that senior developers started spending more time in actual production after the staff reduction. The remedies discussed in this paper were mainly taken during Project NOR1 first five months of execution.

Similarly, the attitudes of key people to communication and onshore/offshore relationship were considered carefully and led to the replacement of the onshore Project manager (with a different onshore project manager who was more communication and motivation-oriented) and the onshore technical architect as described in the previous section. These people were selected for their attitudes and standing in either onshore or offshore organization as much as for the technical skills.

Some specific necessary attitudes we identified:

- peer attitude between onshore and offshore operations
- high level of fluency in English (or the common language between onshore and offshore)
- high attention to communication issues
- low tolerance for communication blocks and resistance to communication
- high personal prioritization and commitment to the project in case of less than 100% allocation to it.

Similarly, for Project NOR2, the inability of the appointed offshore coordinator to dedicate enough energy and time to the project was identified as a major issue and led to the onshore Project manager taking on his duties as described in the previous section. In this case was neither a matter of skill level or unwillingness, but simply of prioritization of available time—and perhaps personal commitment.

5.3.4.3 Lessons Learned

While again it is hard to infer general rules, our current working hypothesis is that agile development assumes all the participants in a Scrum team to be both technically skilled, and willing and able to establish strong communication between them. With high productivity expectations and short iterations, the attitude, time availability or skill level of individual members can quickly result in lower than expected results for entire team, concretely visible in burndown charts and progress reports to the steering group.

Finally, proactive and energetic management actions are essential to correct the issues that risked delaying or compromising the two projects. A tight communication loop between the Project manager (who identifies challenges and proposes corrective actions) and the project owner and a project's steering group (which provide the executive support to the actions) were essential to the quick correction of issues.

5.3.5 Dealing with Idle Time

5.3.5.1 Challenges

An interesting issue that was experienced in both projects was what we came to call the idle time problem. In a localized Scrum effort, clarifications, short discussions and coffee chats about individual tasks or user stories are routinely taken by the project members with personnel from the line organization of the Customer. In an offshore setting, such clarification requests or communications were bound to happen by email, messenger system, videoconference or phone.

Customer's staff, however, had their staff work to do. While the project plan had made allowance for the availability of appropriate personnel on Customer side to clarify things when necessary, no specific rules had been set for response times. As a consequence, and notwithstanding the efforts of the onshore Scrum masters, the response time to such clarification was much longer than would have been experienced if the development team were sitting at the same location. Even Scrum masters at times risked becoming bottlenecks in facilitating communication between offshore developers and Customer's representatives. This resulted in offshore developers sitting idle at times, waiting for feedback from Customer's personnel. Of course, every effort was taken to switch to tasks for which the work could continue, but that lead quickly to a fragmentation of the developers' work which had a sizeable impact on the velocity.

5.3.5.2 Solutions

The challenge was resolved by negotiating triggering rules for which Supplier and Customer agreed to well defined response times for different classes of questions;

whenever response time exceeded the expected one, a certain number of story points were credited to the Supplier, using a story point equivalent in hours. For Project NOR1, this was approximately 40 hrs per story point. The actual value depends of course on which story is used as “unitary” story point and the relative time estimations. The amount of these “idle time” story points was reported at every sprint end together with other measures (done stories, burndown chart, produced story points vs. planned etc.). A major element in the negotiation consisted in getting an agreement on the fragmentation problem—whereas a developer could not simply “find other tasks to work with” and human context switch had a noticeable cost.

Questions were classified, for example, as stopping (i.e. the developer cannot proceed further without an answer), critical (i.e. the developer can continue, but the question will become stopping soon enough), need for completion (i.e. the developer needs an answer before the task or story can be declared done) etc. A practical example of a rule was, for example, to start counting “compensation” story points if a critical question was left unnoticed in the issue tracking system for more than 8 hrs.

We also considered the option of defining a “budget” of questions (e.g., n “stopping” question, m “need to complete” questions, etc.) per sprint (for example, as a function of the total complexity in story points to be delivered in a given sprint). Ultimately we discarded these ideas for simplicity reasons, since we were midway in the project, and because that would have further increased the communication complexity in negotiations—with a Customer who was already experiencing the challenges of adopting agile thinking up to a contractual level.

This provided a clear incentive to the Customer organization to prioritize clarification work and a clear measurement of the consequences of communication inefficiency. At the end of the project, idle time compensation constituted about 5% of the total amount of story points credited to the Supplier.

Interestingly, even if the midproject negotiation with the Customer for introducing the rules (and thus make them accountable for their part in potential delays) was challenging and time-consuming on the business management side, the main challenge encountered by the project manager in implementing the solution was in establishing enough discipline in our own team to report and register questions in a timely matter, so that the corresponding accounting could be performed. The technical implementation of the rules in the issue tracking system was trivial, and represented a minimal overhead.

5.3.5.3 Lessons Learned

The general lesson in this case is to provide explicit triggering rules and a compensation mechanism as part of the initial contract. In an agile spirit, this stimulates collaboration and risk sharing between the delivery and the receiver organizations on communication delays and ensures that this is not seen as a problem of the delivery teams only. Once done in the initial contract, and the corresponding setup of the issue tracking system is performed, the management- and technical implementation-level problems are practically eliminated.

A classic obstacle at the negotiation table is the idea that developers do not need be “idle” since there are other tasks to which they could dedicate their attention while waiting for information. The counter-argument here is simple: one of the reasons for adopting agile methodology is high productivity, and this view is usually shared by the Customer (which otherwise wouldn’t have agreed to an agile/distributed model in the first place); therefore a practical way to illustrate the problems with frequent human context-switching (and thus achieve an agreement on the need for prompt response from the customer, which in turn leads to agreeing on triggering rules) is to provide a simple single-slide example collected from actual time-reporting or observation, showing how much actual developer productivity suffers as a consequence of he/she constantly “picking up a different task”. This worked well in our case, and agreement was ultimately reached.

Finally, an essential tip is not to consider this only a contractual or technical implementation challenge, but also one internal to the team. Initially, the developers did not focus very much on accurately reporting their questions. The project manager engaged in explicit, repeated communication and follow up with them, so that they actually perform the necessary reporting, but that was not enough.

We identified as main reason the fact that, while Scrum is a very disciplined method, none of its practices actually require reporting about communication with customer people—in a local context, such communication is supposed to just happen. Therefore, developers tend to resist additional disciplines unless they are explained extensively and integrated with the rest of the Scrum principles and routines. In practice, adding a short question about pending questions to the daily scrum meeting (and checking their presence in the issue tracking system) proved to be an efficient way to clarify to the team that the activity was as necessary as, for example, daily task follow-up. The cost was a little longer stand-up meeting (about 20 minutes) but proved to be worth its value.

5.3.6 Achieving Motivation and Peer Feeling

5.3.6.1 Challenges

One of the reasons for the existence of global delivery of software or services is obviously cost reduction, achieved via exploiting cost differentials between countries—an exploit whose cost is, in turn, made negligible by the widespread, global availability of tele- and data-communication infrastructure.

However, it should be already clear to the reader that having a lower cost is hardly a motivating factor for most software engineers; rather the opposite. They are traditionally motivated by their expertise, their ability to perform complex tasks, understand customer requirements, deliver on time, mutual respect and recognition and so forth (see for example [3]).

5.3.6.2 Solutions

As a consequence, an issue which was explicitly addressed in both projects was the establishment of a peer relationship between onshore and offshore organization. We made a point to emphasize that cost reduction did not correspond to value reduction, and acted consequently (in both daily project work, attention to individuals' attitudes, ritual events—such as milestone meetings and delivery parties, rewards, and the project manager attitude, for both projects), as allowed by the fact that onshore and offshore were separate line organizations, with their own rules and expectations.

In surveys taken with the offshore personnel at the end of the project, this was reported as a major factor for the motivation and commitment of the offshore team.

5.3.6.3 Lessons Learned

This challenge exists, but it may be considered impolite to address it directly. Our tip is to explicitly state the peer status of all components of the team and ensure that project manager, technical architect and all other personnel do share the peer attitude and values; and then take consistently that view in any pertinent practical decision. Creating and fostering a culture of mutual respect (by both statements and actions) is invaluable when your delivery, credibility and success are—literally—in the hands of an offshore operation.

5.3.7 Adapting Governance and Steering

A major issue emerged in the initial phases of both Project NOR1 and Project NOR2, related to the expectations and language used in steering group meetings and in taking steering decisions; and it was an issue of mindset.

5.3.7.1 Challenges

At that point in time, some top officials at the Supplier onshore organization thought of global delivery mainly as a mechanism for specification-driven work. In brief, where complete specifications are shipped and deliverables are returned. The very language and expectations in the Supplier internal steering group—including the processes and measurement in the quality/business systems—were heavily influenced by such mindset. This included items like progress and revenue accounting (reported in burndown charts and done stories on the side, where hours used vs. estimated were used for internal accounting), expectation of reports on hours spent, unfamiliarity with the concept of “done” stories and so forth.

5.3.7.2 Solutions

The language and expectations barrier between some management and the project team was recognized early by the project owner, who took a large role in translating the language of the project into the language used by the rest of the steering group. Scrum or agile-specific wordings (“user story”, “story point”, the notion or “done” or not “done” stories, “burndown charts” etc.) were initially foreign to such group, and a vast amount of effort was spent filling the gap and establishing a language and expectations platform on which effective decision-making could be done.

In hindsight, it is probably more efficient to establish such platform beforehand, by opportune theoretical and case-based training. Currently, the Supplier is executing a program of high-level training for sales and management officials about agile methods (both in conjunction with offshore distribution or not).

5.3.7.3 Lessons Learned

Adopting agile processes can be a challenging and transformative process in both Customer and Supplier; agile/distributed project add even one more layer of complexity; introducing them need to be seen as a change process—that is, assuming that language, jargon and even agile concepts are not widespread in the organization. Everyone participating in a steering group, including the project manager(s), should be aware of this and adapt their communication consequently.

Furthermore, as any change process within a delivery organization, successful change requires a management sponsor, with the understanding, authority and commitment to act as a bridge and change agent in the appropriate layers of the organization. Having such a sponsor is a very strong success factor for introducing agile deliveries, and was an essential factor in the success of both Project NOR1 and NOR2.

5.4 Conclusions

Both distributed development efforts and agile practices are becoming widespread and have entered, or are entering, the mainstream. However, their combination is relatively new, and this is evident in the kind of challenges and issues we met—from contracts, processes and mindsets tailored for traditional processes, expectations of “defined” processes or a less-than-collaborative attitude based purely on cost accounting.

The benefits and rewards of the agile/distributed model are, however, potentially immense: the productivity, high-value and efficiency of agile processes together with the scalability, cost benefits and size allowed by distribution of delivery. Intangible benefits also include the opportunity for software engineers from disparate places to work together in the intense way that is proper of agile efforts, with the corresponding learning, improvement and ultimately greater quality results.

For example, in the projects post-mortems we evaluated that realizing equivalent functionality with agile projects in a non-distributed way (i.e. with only local people) would have cost to the Customers about three and a half times and, on top, required several distinct projects with the consequent administrative costs; and that specification-driven distributed efforts would have most likely failed.

Of course, the transition from traditional, distributed or agile models to the agile/distributed one is—like most substantial changes—neither easy nor free of charge. It requires investment, dedication, hard work, an open mindset and on occasion, a bit of luck.

An appreciation of the differences and the problems challenges to the combinations—some of which we have illustrated in this chapter—may make a lot of difference in the final result. However, Project NOR1 and NOR2 are a testimonial to the fact that the transition can be achieved in the real world: they were actively steered and ultimately successful. And even if the lessons which we have drawn from their execution will need further validation and will certainly be improved or surpassed, we hope that sharing them will contribute to further successes in similar endeavors.

References

1. Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum*. Upper Saddle River: Prentice Hall PTR.
2. Sutherland, J., Schoonheim, G., Rustenburg, E., & Rijk, M. (2008). Fully distributed scrum: The secret sauce for hyperproductive offshored development teams. In *Proceedings of the int. conf. AGILE* (pp. 339–344).
3. DeMarco, T., & Lister, T. (1999) *Peopleware: Productive projects and teams*. New York: Dorset House.

Chapter 6

Onshore and Offshore Outsourcing with Agility: Lessons Learned

Clifton Kussmaul

Abstract This chapter reflects on case study based an agile distributed project that ran for approximately three years (from spring 2003 to spring 2006). The project involved (a) a customer organization with key personnel distributed across the US, developing an application with rapidly changing requirements; (b) onshore consultants with expertise in project management, development processes, offshoring, and relevant technologies; and (c) an external offsite development team in a CMM-5 organization in southern India. This chapter is based on surveys and discussions with multiple participants. The several years since the project was completed allow greater perspective on both the strengths and weaknesses, since the participants can reflect on the entire life of the project, and compare it to subsequent experiences. Our findings emphasize the potential for agile project management in distributed software development, and the importance of people and interactions, taking many small steps to find and correct errors, and matching the structures of the project and product to support implementation of agility.

6.1 Introduction

Globalization and other competitive factors continue to push organizations to operate more effectively and efficiently. Distributed development continues to become more common, as a result of increased outsourcing and offshoring, as well as the expansion of free and open source software development. As organizations and individuals continue to gain experience with distributed development models, they search for ways to maximize the benefits and minimize risks. Although disciplined processes have advantages in some distributed projects, agile processes have a great

C. Kussmaul (✉)
Muhlenberg College, Allentown, PA, USA
e-mail: clif@kussmaul.org

potential in others, particularly when there are frequent requirement changes and other dynamic factors. Thus, some practitioners and researchers are investigating the applicability of agile practices in distributed environments.

This chapter describes lessons learned from a three years long agile distributed project that involved two development centers and key people at several other locations. The focus of investigation was to retrospectively capture and learn from the experience emphasizing the following aspects:

- Things that went well in the project (highlights, best practices, etc.)
- Things that did not go well in the project (errors, lessons learned, etc.)
- Dos and Don'ts

The information was obtained from interviews with a variety of participants several years after the project finished, and from further conversations with some of them to clarify or expand upon issues discussed in the responses.

The next section summarizes the experience, including customer context, project organization, how agility was introduced, project activities, and the evolving relationship. The following section discusses lessons learned and relevant issues involving people, processes, and coordination.

6.2 Case Overview

6.2.1 Background

The studied project was initiated by the customer company—a privately funded software product company based in the mid-Atlantic region of the USA. Most employees of the company work in a development and operations center, or a smaller sales and marketing office. Over several years, the company had invested significant resources to develop a powerful and flexible data processing engine, with a strong development team, and an extensive feature roadmap. Since the engine was a strategic asset, the customer planned to keep most core development activities in-house. However, the engine required expertise and effort to configure, and so the company planned to use it in a series of front-end applications that were more user-friendly, and domain-specific. These applications presented several challenges:

- Their system requirements could change frequently in response to external market conditions, customers, and competitors. Some of the front-ends involved external standards that were not yet stable, and could change significantly from month to month. Time-to-market was also critical.
- They could require intensive domain analysis phases, and were targeted at business users, not technical users. Thus, their analysis, design, and development required different knowledge and skills than the engine.
- The number and intensity of these application efforts would vary over time, which presented staffing challenges. Diverting the customer's development organization to work on the front-ends would adversely affect the engine.

Table 6.1 Customer company

USAsoftware	
Number of developers	5–20 (over project life)
When was agile introduced	2003 (start of the project)
Domain	B2B software

Table 6.2 Onshore supplier company

USAconsult	
Number of developers	3–5 (over project life)
When was agile introduced	2002 (founding of the company)
Domain	software development

Table 6.3 Offshore supplier company

INDdevelop	
Number of developers	approximately 300
When was agile introduced	2003 (start of the project)
Domain	software development

- Cash flow was a concern, since the customer was already supporting a significant development effort for the engine.
- The engine was under active development, with capabilities added regularly, so the boundaries and interfaces between the applications and the engine would change over time.

Thus, the customer decided to hire an external supplier to develop the first application. Important factors in selecting the supplier were: the technical and interpersonal abilities of the supplier’s consultants and their proximity (onshore location within a driving distance of the customer’s development center and primary sales office). In addition, the supplier also had experience in managing off-site and off-shore teams and an existing subcontracting relationship with a CMM-5 organization in southern India.

So, the studied project was spread across three companies: the customer and the subcontracted supplier chain (see Tables 6.1, 6.2, and 6.3).

6.2.2 Project Organization

The project involved three teams, which will be referred to as: off-site, on-site and planning (see Tables 6.5, 6.6, and 6.7).

The *off-site team* (in India) focused on the graphical user interface (GUI) and database, including low-level design, implementation, and testing, but was also involved in analysis and design, and reviewed design documents and proposals. The

Table 6.4 Project overview

Project	
Duration	3 years
Status	finished
Agile practices	adapted from Scrum and Feature-Driven Development
Involved locations	USA, USA, India

Table 6.5 Off-site team

Locations	Number of members	Roles
India—INDdevelop	5–16	1 lead, developers & testers (low-level design, implementation, and testing)

Table 6.6 On-site team

Locations	Number of members	Roles
USA—USAsoftware	3–6	1 lead, 2–5 developers (integration between database and data processing engine)

Table 6.7 Planning team

Locations	Number of members	Roles
India—INDdevelop	1	off-site project lead
USA—USAsoftware	2	on-site team leader VP for software development
USA—USAsoftware	1	VP for product management
USA—USAconsult	2	consultants

on-site team (of customer employees) focused on integration between the database and the data processing engine. At the beginning of the project, the customer’s development group had quite informal processes, which became somewhat more formal over time. The *planning team* focused on analysis, GUI mockups, high-level design, coordination, and some testing, and resolved any issues that extended beyond one of other teams. It was a virtual team, consisting of onshore consultants (who typically spent one or two days per week on-site at the customer), the leaders of the other two teams, and the customer VPs for project management and software development. There is an extensive literature on virtual teams (e.g. [4, 8]; see also Further Reading). The off-site team ranged from five to sixteen full time people,

and the on-site team ranged from three to six full time people. Most planning team members had other responsibilities, often outside of the project; total effort for the planning team was typically the equivalent of two to three people (see also Fig. 6.1). The on-site and planning teams also interacted with the in-house developers working on the data processing engine, although the latter were not directly involved in this project.

6.2.3 Introduction of Agility

The relationship began with one hourly consultant, who spent two to three days a week on-site at the customer site over several months. The consultant studied the application domain, analyzed requirements, developed a high-level design and a list of potential features, and supported in-house development of a proof-of-concept and demo. Based on the confidence and trust developed through these initial deliverables, the customer decided to outsource most of the initial application development to the supplier and the off-site team.

The challenges described above made it difficult to accurately estimate the scope or resource requirements of the project. Furthermore, the customer might need to cancel the project as more was learned about the project scope, resource requirements, and commercial potential. Thus, a traditional disciplined process was considered too risky, and an agile project structure was proposed. The onshore consultants had studied and used some agile processes, and recommended that the project adapt Scrum [9] and Feature-Driven Development (FDD) [7] to help manage the risk and uncertainty. The customer readily agreed with this approach, while the off-site team was more reluctant, since they were used to CMM-5 processes and most were unfamiliar with agile approaches. Thus, the participants decided to focus on agile project management to help coordinate activities between the off-site and on-site teams, and not on agile development practices (such as XP). The participants also decided to emphasize the spirit of these agile processes rather than the details, since several of the participants had previous experience managing and adjusting development processes, and since Scrum and FDD were designed primarily for co-located teams.

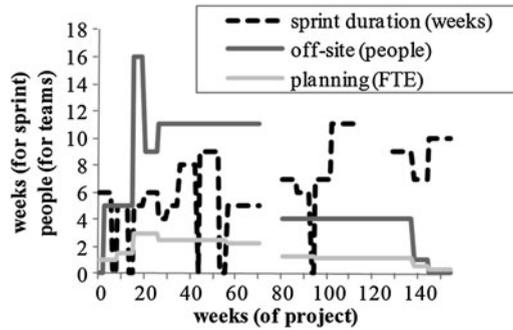
The next section describes key project activities.

6.2.4 Overview of Project Activities

Early in the project, the planning team developed an initial feature list (or product backlog) and the overall product architecture (or system model). This feature list was used to estimate the range of likely project costs, and both the list and the estimate were updated at intervals throughout the project.

The project was divided into time-boxed sprints of four to six weeks. As the project evolved the teams grew and some sprints became as long as twelve weeks,

Fig. 6.1 Planning team size (in FTE), offshore team size (in people), and sprint duration (in weeks) over the 160 weeks project duration



in part to reduce the testing and documentation overhead for each delivery, although more time was spent determining the scope of each sprint. Towards the end of the project, sprints became shorter and the off-site team shrank, as the customer took ownership of the project.

Before each sprint, the planning team developed and signed a formal proposal that identified the major milestones, the sprint feature list, and a price range. During each sprint the teams implemented features from the sprint feature list. After each sprint, the supplier determined the final price (within the given range) based on actual effort and the scope of work completed. This allowed the planning and off-site teams to be more responsive to uncertainty and change requests.

Figure 6.1 summarizes the effort for the planning and off-site teams, and the sprint durations, over the 160 weeks of the project. (Effort for the on-site team is not shown, because it was not formally tracked during the project.) The planning effort grew gradually over the first 20 weeks, in part due to transition periods associated with some staffing changes involving the onshore consultants. The off-site team started with five people, expanded to eleven through the middle of the project, and dropped to four for the second half of the project. The off-site team had sixteen people for one sprint, but then the planning team realized that some work could be done more effectively in-house, and reduced the off-site team accordingly. This illustrates how an agile process enabled the project to adjust staffing promptly to reflect project needs. Note that some data was unavailable, and that occasionally there was a delay between the end of one sprint and the start of the next, shown by the duration dropping to zero briefly.

The on-site team's work generally paralleled the sprints, but was less formal, since the team members were salaried employees of the customer. This shifted some as the project evolved and the teams' responsibilities became more coupled. In retrospect, it might have been better to use the same processes, including written proposals before each sprint, for both the off-site and on-site teams. This would have emphasized the shared goals and responsibilities, and avoided some problems where the on-site team was late in completing work that the off-site team needed to complete a sprint.

The three teams used multiple coordination techniques:

- A shared mailing list to archive ongoing communication and discussion. Teams sent occasional status messages to the list, describing significant changes, current problems, and any questions. At times these messages were daily (at the end of their respective work days), at other times they were less frequent, particularly when there was less coupling between the team activities.
- The off-site team had daily meetings to review status and address major problems that prevented the project from moving forward. These meetings were usually conducted via instant messaging or phone with an onshore consultant on the planning team, and lasted around fifteen minutes, though they occasionally lasted longer. They were usually late afternoon in India and early morning in the US, but sometimes early morning in India and late evening in the US. In retrospect, the on-site team and an onshore consultant should also have had daily meetings, to improve coordination between the teams.
- A CVS repository stored all requirements, designs, source code, and related documents. The code was kept in a working state at all times, with any exceptions noted via email, so that everyone could see the real progress. In effect, there were daily code deliveries between the teams.

6.2.5 Cross-border Relationship Dynamics

Cross-border relationships can be viewed along two dimensions [5]. *Dependence* indicates how much ownership and control is transferred to the partner. *Strategic impact* indicates how much the relationship affects competitive positioning and long-term strategy. As a relationship increases in either of these dimensions, collaboration becomes more critical. The following are different options that organizations can consider when initiating a cross-border relationship:

- *Support* relationships (low dependence, low impact) simplify internal operations, such as payroll processing, basic IT functions, or custodial services. They are used selectively, easily monitored, and easily changed.
- *Alignment* relationships (low dependence, high impact) are also used selectively, but usually for more strategic purposes, such as expert consulting, or specific projects.
- *Reliance* relationships (high dependence, low impact) are usually used to reduce costs on a broader scale and for longer time periods.
- *Alliance* relationships (high dependence, high impact) usually develop from other types of relationships. These are uncertain and dynamic strategic relationships, and require trust, communication, and, often, shared goals or incentives. The key success factor is “a mutual understanding between clients [customers] and their service providers [suppliers]” [5, pg. 92].

Over time, the customer-supplier relationship evolved through several stages. Initially, it was *support*, with an hourly consultant to study and evaluate specific application domains. It then evolved to *alignment*, where the consultant helped define

requirements and a high-level architecture for a specific front end. Once a proof-of-concept was completed, on-site staff and a small off-site team began to design, implement, and test. The relationship then shifted toward *reliance* as the off-site team expanded, the customer developed confidence, the market opportunities became clearer, and the scope of the overall project increased. Finally, the relationship evolved toward an *alliance*, where customer and supplier worked together to identify future directions. As the application matured, the relationship shifted back toward alignment as the off-site team was reduced and the customer took over ongoing system maintenance.

Most participants adapted quickly to the agile approach. Both development teams could continue using familiar development activities (CMM or informal); the agile project management helped them understand current priorities and directions, and enabled them to contribute to planning future iterations. Team leaders and managers could track team progress, see working code on a regular basis, avoid conflict over scope and resources. The customer's testing and documentation groups liked having access to working code so they could monitor the project and plan their activities accordingly. As described below, agility also improved relationships with other stakeholders, including sales and marketing.

6.3 Lessons Learned

Many of the issues, problems, and lessons learned that were described by participants involved familiar themes addressed in the literature. This case study is especially interesting because the studied project involved three different locations and multiple outsourcing relationships.

The lessons learned are divided under three topics: *people*, *process*, and *coordination*. Some of the offered findings and practical tips address challenges related to distribution; others address challenges related to agility. One practice, however, deserves a special attention, as it is a prerequisite for initiating a good collaboration.

Practical Tip: Focus on win-win aspects wherever possible to prevent contracts and conflicts from disrupting the project.

In this project, each sprint had a separate proposal, which specified a price range to accommodate incomplete and changing requirements, and to avoid having to finalize all requirements. As a result, both customer and supplier sought to minimize costs, but the supplier could adjust the final cost if there were requirement changes or implementation problems. This helped to avoid the conflicts inherent in fixed-price contracts, where (rationally) requirements should be interpreted broadly by the customer and narrowly by the supplier. At a more personal level, the project's scope and benefits were explained to the customer's employees, so they were confident that they would not be replaced by the supplier, and that the supplier was

enabling the customer to address new opportunities. In fact, the customer's development group grew by several hundred percent during the course of the project. In this project, maintaining win-win relationships was easier because most of the key people had been both customers and suppliers of software development, and thus they were familiar with each other's roles and challenges.

6.3.1 People

The first set of lessons learned focus on people and interactions between them, and particularly some of the challenges of starting and maintaining relationships.

Practical Tip: Ease participants into relationships with remote teams, and arrange regular and extended visits between locations when feasible.

Many key participants had previous experience with distributed teams. For example, several members of the off-site team in India had worked in the US. The onshore consultants had all spent time in India, had worked with distributed teams before, and typically spent at least one day a week on-site. The customer was also familiar with distributed teams and supporting tools. There were also occasional teleconferences (usually for training) between the planning team, the on-site team, and the off-site team. Nevertheless, there were some communication problems early in the project. For example:

... during the first episode, [the planning staff] tried to 'micro-manage' the team here. Assigning daily tasks and tracking. It was not well received at our side. ... Especially when your customer is faceless, there is this definite overhead of trying to understand his requirement in addition to the usual pressure of quickly producing results. [Off-site Team Leader]

In retrospect, this previous experience was important but not sufficient. Some of the off-site team should have visited the customer, and/or been visited by an onshore consultant or customer employee to communicate requirements and develop better relationships.

The cost of such travel was a concern, as it is in many projects. In particular, this project was organized into sprints, and there was a real probability that the project could end after only a few sprints. This made it more difficult to justify such expenses, as compared to a project with a known scope and duration. Thus, this is a possible disadvantage of this approach to agile distributed development.

Practical Tip: Be sensitive to cultural differences, especially between organizations and between regions or countries, including differences in how people interact with each other and resolve problems.

The cultural awareness is especially important in agile distributed projects, where culturally diverse team members are required to form a cohesive project team. Cultural differences in the investigated project were exhibited on various levels. For example, Indian national and corporate culture is accustomed to centralized, top-down control, and may view direct questions as a challenge to authority. The off-site team was accustomed to static requirements and disciplined processes, while the on-site team had very informal processes. At times it was difficult to convince each team to explore other approaches.

In this project, the participants minimized cultural differences between the off-site and on-site teams because most of their interactions went through people who were used to working across cultures.

Practical Tip: Maintain a good attitude throughout the project, and particularly at the end.

Too often, projects start with energy and enthusiasm, but end with frustration, regrets, or resentment. In many cases, these problems have grown throughout the project, and leave everyone with bitter memories. This project started with the expectation that the customer would take over the maintenance of the code, and the supplier and customer worked together to make a smooth transition. Therefore, in contrast to many other outsourcing collaborations, the suppliers in this project were involved in the challenging work, while the maintenance tasks, often regarded as boring, were actually back-sourced.

...once the initial magic of new ‘development’ is gone, people hate to maintain. . . . But in this project, once the objective was met, a transition phase was planned where [the customer] took back the code and decided to maintain it. I have not seen such happy endings. [Off-site Team Leader]

Thus, we need to recognize that “the major problems of our work are not so much technological as sociological” [3, pg. 4] (original emphasis); in other words, the major benefits and risks are tied to people and relationships rather than to tools and techniques.

6.3.2 Processes

The second set of lessons learned focus on process, and the importance of selecting and understanding appropriate projects and processes.

Practical Tip: Avoid projects that are too small to amortize the overhead required for an effective distributed team.

Very small projects are more suitable for co-located teams, unless a distributed team is cohesive and already has experience working together. The author has

worked on agile distributed projects with as few as three developers, but these projects have been less successful. In the current project, some of the initial explorations could have been completed more efficiently by an in-house, co-located team, but were assigned to a distributed team in anticipation of subsequent phases of the project.

Practical Tip: Start with a small distributed team and grow it over to meet project needs.

This may appear to contradict the previous tip, but there is a difference. A small project that expects to grow can invest time to develop relationships, processes, and practices that will be valuable later, even though they may be inefficient initially. Starting small allows participants to work closely and adapt more quickly when problems occur.

Practical Tip: Explain agile philosophy, not just agile practices.

It is widely recognized that agility is a philosophy, not just a set of processes to be followed. When the consultants recommended that the customer adopt an agile process for the project, they explained how it would help manage risks and uncertainty, and the customer readily agreed. There was also a shift during the project. At first, some participants, particularly from customer's sales and marketing organization, pushed hard to have some feature added to the product, partly because they were accustomed to long development cycles and long delays in adding such features. When these demands occurred, the planning team explained that requirements for the current sprint were frozen, but that a new sprint would start in a few weeks and finish a few weeks later, and that any urgent features could be considered and included. As the sales and marketing people became accustomed to this approach, they recognized that they didn't have to push so hard to get features in the pipeline because the project could adjust more quickly to changing market conditions.

The customer's development group had an informal development process, which may have made them more open to agile. A customer that followed more formal processes might have been reluctant to experiment with agile approaches in 2003, when agile was less well known. Interestingly, the off-site team was more reluctant, since they were familiar with CMM-5 processes and mostly unfamiliar with agile approaches. Fortunately, their CMM procedures were flexible enough to accommodate agility on the project management level. However, it might have been more difficult for them to adopt agile engineering practices (e.g. XP), since it would have required many more changes in their day-to-day activities and processes. Similarly, it would have been difficult for the customer's development group to quickly adopt CMM style processes. It is quite feasible, and sometimes preferable, for teams to use different methodologies that reflect different cultures and requirements. In this project, the agile processes bridged these different approaches by providing a com-

mon management framework which allowed each team to continue using familiar development practices.

Nevertheless, during the project the off-site team came to understand some of the benefits and risks of agile processes, as evidenced by the following quotations:

If I am going to work on anything small like less than 20 people, I would seriously pursue agile . . . if I have a choice. Because I actually saw it in action. Till you see it, you don't really trust it's possible at a practical level. [Off-site Team Leader]

I am little worried whether this methodology or process will suit for a big system with lot of modules and lot of developers (more than 15). In that case, we may have to proceed with requirement analysis, then architecture design and thus creating the module level picture and its interfaces. I think the module development can follow the agile methodology. [Off-site Team Member]

Our process recommendations can be summarized in words attributed to Tom Peters, “test fast, fail fast, adjust fast”. Start with small teams and projects, use sprints or other short iterations, and provide multiple opportunities to find and correct problems before they become larger and more serious.

6.3.3 Coordination

The third set of lessons focus on the coordination between the teams.

Practical Tip: Keep requirements analysis, research, and architecture close to the customer, at least initially.

For example, during the first few sprints the planning team developed the core architecture and a set of practices to serve as guidelines, particularly as the off-site team evolved and became more familiar with the project and application domain. As the project and team grew, the planning team became less involved in coding, and more involved in planning, reviews, and other coordination to serve as a bridge between the off-site team and the customer. Similarly, as the off-site team gained experience, they became a more important part of subsequent planning, since they were most familiar with the working system.

Practical Tip: Use key documents to bootstrap the project by establishing a common framework for distributed team members and other stakeholders.

Brooks [1] calls this the *documentary hypothesis*: “a small number of documents become the critical pivots around which every project’s management revolves”. Agile projects generally require less documentation than disciplined projects, but distributed projects generally require more documentation than co-located projects, since informal communication is more difficult. For example, questions that can be answered within seconds in a co-located team can take hours or even days to answer in a distributed team. This project depended on the following documents:

- The master feature list—every completed, scheduled, or envisioned feature—and the corresponding estimated budget for the entire project.
- A detailed data model and data dictionary, which were key interfaces between teams and components, including the interface to the software engine.
- Requirements and designs for subsystems in the current or next sprint. These documents were usually discarded (i.e. not maintained) after the corresponding features were developed and tested.
- A list of tasks and related information for the current sprint.

Practical Tip: Coordinate carefully, and deliver working software early and often, so as to respond quickly to the necessary changes.

For example, by focusing on activities such as analysis, customer interaction, and testing, the planning team was able to send requirements to the off-site team in the evening, and have the resulting changes implemented the next morning. Because each team was eager to start work when the other finished for the day, this approach also discouraged excessive overtime. This round-the-clock cycle can work well for activities such as analysis and testing, but generally works less well for other development activities. In addition, this close coordination can be stressful and tiring, so often there would be looser coupling in the middle of a sprint, and then tighter coupling towards the end.

Frequent deliveries gave the customer many opportunities to review the project status and make appropriate changes, especially for user interfaces. It provided effective communication between all stakeholders, and built confidence and trust among the teams. Planning was still important, although it took less time than many outsourced or distributed projects. Each sprint was a separate proposal and contract, and included planning for the next sprint.

Practical Tip: Define responsibilities, interfaces, and supporting processes *between* organizations, key individuals, and teams.

As in any technical design, seek to minimize coupling and communication between components and teams; also seek to maximize cohesion so each group can focus on particular objectives. For example, the planning team decided to use a database as the primary interface between the front end application and the data processing engine, in order to reduce coupling and minimize the impact of changes in the engine. Similarly, it was often beneficial to have a designated liaison between the teams, to provide conceptual integrity and prevent miscommunication. Thus, members of the planning team coordinated activities between the in-house and off-shore teams.

In this project, the partitioning worked well:

The roles were correctly evolved. [Planning staff] played the role of architects and shaped a wonderful architecture. I mean [they] were able to understand the real objective of the

architecture. The need to generate a lot of screens, fully flexible and code generated because the domain . . . was being very dynamic. I think [they] achieved that admirably. And [the off-site team] was perfect to ‘assist’ . . . in realizing it. [The off-site team] had matured enough by then to say no to quick fixes and cowboy programming to add features without compromising the ideals of the architecture. . . . we were able to handle the code reasonably well. So I guess the project was blessed to have ‘real’ architects and some senior programmers from our side. [Off-site Team Leader]

Our coordination recommendations can be summed up by Conway [2] who observed that the structure of an organization determines (or at least biases) the structure of systems it creates. This is particularly relevant in agile distributed projects, where the boundaries between organizational units are more pronounced, so that agile interactions are stronger within co-located units than between units.

6.4 Conclusions

This chapter describes an agile distributed project involving multiple organizations and relationships, and expands an earlier report [6]. Our findings yield a variety of best practices and useful lessons learned, including:

- Focus on win-win aspects to minimize potential disruptions.
- Ease participants into relationships with remote teams.
- Be sensitive to cultural differences, and use key people to bridge gaps.
- Maintain a good attitude throughout the project, and particularly at the end.
- Avoid projects that are too small to amortize overhead.
- Start with a small distributed team and grow it over time.
- Explain agile philosophies, not just agile practices.
- Keep requirements analysis, research, and architecture close to the customer.
- Use key documents to bootstrap and provide a common framework.
- Coordinate carefully to allow distributed teams to respond quickly to changes.
- Deliver working software early and often to build confidence and trust.
- Define responsibilities, interfaces, and supporting processes between teams.

Many of these practices can be summarized in three key ideas:

- “The major problems of our work are not so much technological as sociological”
—Tom DeMarco and Tim Lister
- “Test fast, fail fast, adjust fast”—Tom Peters
- The structure of an organization determines (or at least biases) the structure of systems it creates—Melvin Conway

The first two ideas are consistent with established agile principles, The third is less familiar, perhaps because traditional agile projects have simple structures. For agile distributed projects, however, this is a key insight which can enable participants in different locations to focus on different parts of the problem.

Overall, this project was a success. The system was completed, despite initial challenges, frequently changing requirements, and a variety of technical issues. Participants in all three project teams were generally positive about the experience, and

learned about the advantages and disadvantages of some agile practices, so they are better able to choose or adapt such practices in the future.

We hope that the lessons described above can help other organizations to work more effectively and efficiently.

References

1. Brooks, F. (1995). *The mythical man-month*. Boston: Addison-Wesley.
2. Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28–31.
3. DeMarco, T., & Lister, T. (1999). *Peopleware: productive projects and teams*. New York: Dorset House.
4. Duarte, D. L., & Snyder, N. T., (2006). *Mastering virtual teams*. Hoboken: Jossey-Bass.
5. Kishore, R., Rao, H. R., Nam, K., Rajagopalan, S., Chaudhury, A. (2003). A relationship perspective on IT outsourcing. *Communications of the ACM*, 46(12), 87–92.
6. Kussmaul, C., Jack, R., & Sponsler, B., (2004). Outsourcing and offshoring with agility: A case study. In C. Zannier et al. (Eds.), *Extreme programming and agile methods—XP/agile universe* (pp. 147–154). Berlin: Springer.
7. Palmer, S. R., & Felsing, J. M. (2002). *A practical guide to feature-driven development*. Upper Saddle River: Prentice Hall PTR.
8. Pinsonneault, A., & Caya, O. (2005). Virtual teams: What we know, what we don't know. *International Journal of e-Collaboration*, 1(3), 1–16.
9. Schwaber, K., & Beedle, M. (2001). *Agile software development with Scrum*. Upper Saddle River: Prentice Hall PTR.

Further Reading

10. Anderson, D. (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. Upper Saddle River: Prentice Hall PTR.
11. Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: a guide for the perplexed*. Boston: Addison-Wesley.
12. Braithwaite, K., & Joyce, T. (2005) XP expanded: Distributed extreme programming. In *Extreme programming and agile processes in software engineering*, Berlin: Springer.
13. CIO Insight (2003). Research: Outsourcing: How well are you managing your partners? 1(33), 75–85.
14. Cockburn, A. (2003). *Agile software development*. Boston: Addison Wesley.
15. Drummond, B.S., & Unson, J.F., (2008). Yahoo! distributed agile: Notes from the world over. In *Proceedings of agile 2008* (pp. 315–321). Washington: IEEE Computer Society.
16. Highsmith, J. (2002). *Agile software development ecosystems*. Boston: Addison-Wesley.
17. Hole, S., & Moe, N. B. (2008). A case study of coordination in distributed agile software development. In *Software process improvement* (pp. 189–200). Berlin: Springer.
18. Holmström, H., Fitzgerald, R., Ågerfalk, P. J., & Conchuir, E. O. (2006). Agile practices reduce distance in global software development. *Information Systems Management*, 23(3), 7–18.
19. Jarvenpaa, S. L., Knoll, K., & Leidner, D. E. (1998). Is anybody out there? Antecedents of trust in global virtual teams. *Journal of Management Information Systems*, 14(4), 29–48.
20. Jensen, B., & Zilmer, A. (2003). Cross-continent development using Scrum and XP. In: *4th International conference on extreme programming and agile processes in software engineering*. Berlin: Springer.

21. Lander, M. C., Purvis, R. L., McCray, G. E., & Leigh, W. (2004). Trust-building mechanisms utilized in outsourced IS development projects: A case study. *Information and Management*, 41(4), 509–528.
22. Lee, G., DeLone, W., & Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM*, 49(10), 35–40.
23. Martins, L. L., Gilson, L. L., & Maynard, M. T. (2004). Virtual teams: What do we know and where do we go from here? *Journal of Management*, 30(6), 805–835.
24. Mockus, A., Weiss, D.M., (2001). Globalization by chunking: A quantitative approach. *IEEE Software*, 18(2), 30–37.
25. Paasivaara, M., & Lassenius, C. (2004). Using interactive and incremental processes in global software development. In *Proceedings of the international conference on software engineering (ICSE) third international workshop on global software development* (pp. 24–28).
26. Paulk, M. (2001). Extreme programming from a CMM perspective. *IEEE Software*, 18(6), 19–26.
27. Powell, A., Piccoli, G., & Ives, B. (2004). Virtual teams: A review of current literature and directions for future research. *ACM SIGMIS Database*, 35(1), 6–36.
28. Ramesh, B., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile? *Communications of the ACM*, 49(10), 41–46.

Chapter 7

Contribution of Agility to Successful Distributed Software Development

Saonee Sarker, Charles L. Munson,
Suprateek Sarker, and Suranjan Chakraborty

Abstract In recent times, both researchers and practitioners have touted agility as the latest innovation in distributed software development (DSD). In spite of this acknowledgement, there is little understanding and evidence surrounding the effect of agility on distributed project success. This chapter reports on a study that examines practitioner views surrounding the relative importance of different sub-types of agility to DSD project success. Preliminary results indicate that practitioners view on-time completion of DSD projects, and effective collaboration amongst stakeholders as the top two criteria of DSD project success, with lower emphasis on within-budget considerations. Among the many agility sub-types examined, people-based agility, communication-based agility, methodological agility, and time-based agility emerged as the most important for practitioners in terms of ensuring DSD project success.

7.1 Introduction

The use of distributed teams, with members situated in multiple locations, to conduct software development tasks in organizations has exponentially increased in recent years (e.g., [9]). Distributed development teams face a number of challenges

S. Sarker (✉) · C.L. Munson
Washington State University, Pullman, WA 99164-4743, USA
e-mail: ssarker@wsu.edu

C.L. Munson
e-mail: munson@wsu.edu

S. Sarker
Copenhagen Business School, Solbjerg Plads 3, Frederiksberg 2000, Denmark
e-mail: sarker@cbs.dk

S. Chakraborty
Towson University, Towson, MD 21252, USA
e-mail: schakraborty@towson.edu

that arise from factors such as geographical and temporal differences, culture, language, varying knowledge levels and a primary reliance on computer-mediated communication [9], making it very difficult to effectively harness such teams. Thus, it is important to examine the factors that affect distributed software development (DSD) team effectiveness.

Recently, yet another “emergent force”, namely, the notion of agility has drawn the attention of researchers and practitioners alike. A growing realization that DSD project teams need to continually react to market dynamics, respond and incorporate changing customer requirements, and deal with technological innovations [4], has made agility an important factor. DSD project managers and software organizations have been quick to adopt “agility principles and approaches” in an effort to render their teams more effective [4, p. 360]. While the promise of higher effectiveness (e.g., [11]), has led to organizations scrambling to adopt agile principles, there is limited evidence of how these principles and approaches contribute to the DSD effectiveness [15]. We thus believe that it is critical to seek the answers to the following questions:

- What are the different indicators of DSD project success?
- What is the relative importance of the different types of agility drivers to the success of DSD teams?

In this chapter we provide a discussion on DSD project success indicators, followed by a general overview of how agility may contribute to distributed organizations. Next, we outline different types of agility and discuss the importance of each type from the practitioner perspective.

7.2 Distributed Project Success

While there is an ongoing discussion about the interpretation of the term ‘project success’, there is a viewpoint that it may be identified by “two concepts” [1, p. 25]:

- Project management success, which refers to the process of the project, especially, the “successful accomplishment of cost, time, quality objectives”;
- Product success, which “deals with the effects of the project’s final product”.

Similarly with regards to success in information systems development (ISD) projects there is no consensus on what the measure should be. Jiang and Klein [8] argue for the focus on on-time delivery and cost (i.e., within budget). Other measures of ISD project success such as the quality of the system being developed, the improvements in the quality of the decision-making, and IS usage, have also been proposed (e.g., [12]). In a more recent study, Chow and Cao [5] assessed the success of agile software projects by examining the time, cost, quality, and scope of the projects. While there is no agreement on the best measures of project success, the above measures represent the *product perspective* or the “hard dimensions” [1, 12]. However, it has been argued that for DSD projects, it is important to assess the *process-based success measures* or “soft” dimensions such as the collaboration effectiveness [1, 9], in addition to the product-based measures.

In this chapter we rely on selected practitioners to determine the relative importance (or salience) of the three success measures for DSD projects: on-time delivery, within-budget, and effective collaboration.

7.3 Types of Agility

Agility has been widely discussed from different perspectives. In this book the readers will find experiences and recommendations for adopting different and often standalone agile methods and practices. In this chapter we observe agility on an organizational level. Thus, we define agility as the “ability of enterprises to cope with unexpected changes, to survive unprecedented threats from the business environment, and to take advantages of changes as opportunities” [7, p. 147].

The gathered knowledge about agility revealed that it is itself composed of different components. Lui and Piccoli [11, pp. 125–126] propose four different components of information systems agility - technological agility, process agility, people agility, and structure agility. From the perspective of distributed ISD, Lee et al. [10] propose that agility is composed of agile IT strategy, agile IT infrastructure, and agile project management. This provides a good start toward our understanding of agility in distributed ISD teams, however, it must be noted that the focus is on macro issues surrounding DSD. We feel that agility dimensions should also incorporate micro issues such as day-to-day operations, application and the use of ideal methodologies, and management of its resources and relationships, which are crucial for such project teams. Sarker and Sarker [14], takes the above issues into consideration, and proposes DSD agility as being of three broad categories:

- (1) resource-based (agility arising from the DSD team’s access to necessary resources),
- (2) process-based (agility arising from the appropriate management of the process of DSD), and
- (3) linkage-based (agility arising from the nature of the ties between the different DSD team members).

Each of these types can be decomposed further into the following sub-types (see Table 7.1).

7.4 Study Background

The essence of decision-making lies in choosing between alternatives [13]. Such choice in turn needs to be facilitated by a supporting framework or an appropriate “multiple-criteria yardstick” [2, p. 91–92]. Consequently it is important to create a framework for assessing the relative importance of the different agility elements. The Analytic Hierarchy Process (AHP) represents a useful technique that enables

Table 7.1 Definition of the types of agility

Agility Types	Definition
People-related Resource Agility	The speed with which the GSD team is able to acquire appropriately, skilled human resources at each distributed location when necessary, and the capability to gracefully and speedily reconfigure itself on demand, by rotating and shifting different team members (across locations or divisions), or by shifting the “control centers” of the project from one distributed location to another.
Technology-related Resource Agility	The ease with which the DSD team members have access to high quality communication media, and development/testing-related technologies, irrespective of their location.
Methodology-related Process Agility	The ease with which the DSD team is able to a) adapt traditional SDLC to make the ISD process more responsive, b) implement agile methodologies/practices to its project context, and c) deal comfortably with abrupt methodology changes during the project.
Environmental Scanning-related Process Agility	The ability of the DSD team to anticipate, recognize, and react to changes in the project due to changes in the environment.
Work Transition-related Process Agility	The capability of the DSD team to enable seamless transition of work from one location to another.
Time-related Linkage Agility	The capability of the DSD team to collaborate with each other without significant temporal delays.
Culture-related Linkage Agility	The ability of DSD team members to rise above any cultural differences that may exist within the team, and work seamlessly.
Communication-related Linkage Agility	The ability of the DSD team to be aware of the actions/reactions of distributed team-members, utilize cultural difference advantageously (where possible), and maintain continuous “connectivity” with the customers.

decision-makers to assess the relative importance of the different attributes on a particular outcome [2], and therefore direct more focused attention to the alternatives that matter the most to an outcome. This is especially relevant for our study too, while our eight proposed types of agility have been justified on conceptual grounds, to provide more actionable guidance to DSD project managers, it is important to determine which of the types need to be considered to achieve project success.

We therefore applied AHP procedures to generate an understanding of the relative importance of the agility dimensions to project success. Six independent experts from large international organizations were involved as experts in this process. All of the respondents were knowledgeable in distributed software development and had conceptual understanding about agility in the ISD context. The respondents also represent a balanced mix of technical (R1, R2, R4) and managerial roles (R3, R5, R6). Therefore their combined responses could be perceived to be unbiased towards either perspective. Table 7.2 provides a summary of respondent profile.

AHP requires the problem to be structured as a hierarchy. In this hierarchy the top level consists of the “objective of the problem” [2, p. 92]. For our study, this refers to overall DSD project success. The next level comprises of the sub-objectives or

Table 7.2 Respondent profiles

Respondent Number	Affiliation and its HQ	Country Currently Based In	Respondent's Years of Experience with DSD	Respondent's Number of Distributed Projects	National Contexts of Distributed Projects
R1	Infosys (India)	UK, Switzerland	10	10	India with Japan, Singapore, Australia, South Africa, and Europe (including Netherlands, UK, and Switzerland)
R2	Microsoft (US), Sapient (US)	India	8	12	US-India
R3	Intec Telecoms (UK)	India	12	100	US-UK-India, UK-Australia-India-Poland, US-UK-Japan-India
R4	Deloitte Consulting (US)	US	2	2	US-India
R5	Federal Govt. (US)	US	10	4	Different parts of US
R6	Agilent (US)	US	4	4-5	US-India

elements playing a role in achieving the objective(s) (i.e., on-time delivery, within budget, and effective collaboration). Following this level are any criterion variables affecting the higher-level objectives (i.e., the different sub-types of agility) (see Fig. 7.1).

Subsequent to the establishment of the hierarchy, there ensues a “prioritization” process. In this process the experts provide judgments (through a numerical pairwise comparison) about the relative importance of all possible pairs of elements on the final objective. In course of in-depth interviews lasting between 60–90 minutes, subjects provided their numerical pairwise comparison ratings using the traditional AHP rating scale ranging from 1 to 9 (where 1 refers to a situation where both components of a comparison is equally important, and refers to a situation where one component is extremely more important than the other). Based on the pair wise comparisons, the relative importance matrix is created for each set of the comparisons. Scores from the relative importance matrix were then normalized and averaged to

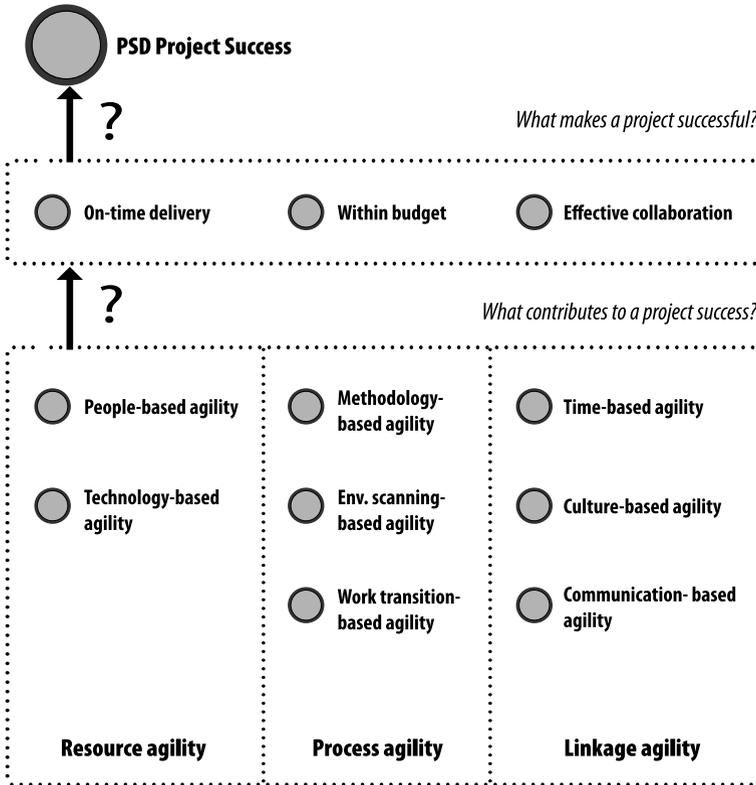


Fig. 7.1 The hierarchy diagram

create relative weights within the comparison set, as well as overall weights (aggregated across the 6 experts) for the entire hierarchy.

7.5 Contribution of Agility to Distributed Project Success

The results of our investigation indicate that DSD practitioners view on-time delivery as the most important success criteria, followed by effective collaboration, and within-budget delivery. The highest ranking of on-time completion is not surprising, given the perception that technology changes at a rapid rate, and if a project is delayed, it runs the risk of the information system in question being obsolete or losing its value for the customer. Given that our respondents were DSD participants, who deal with the challenges of this form of software development (such as communication and coordination across time and space), further justified their view of effective collaboration as a key criterion of DSD project success. As Procaccino et al. [12] argue, practitioners are increasingly valuing the team development process as a key success indicator. The relatively low importance assigned to budget also seems to

Table 7.3 Importance of rankings of agility dimensions

Rank	Agility Dimensions
1	People-based
2	Communication-based
3	Methodological
4	Time-based
5	Technology
6	Work Transition
7	Environmental
8	Cultural

make sense. Too much focus on the cost can result in sort of a tunnel vision, which can be counterproductive for the optimal management of the development process itself. Further the experts also prioritized different types of agility. Table 7.3 displays the ranking of the different types of agility with respect to DSD project success. We can see that the *people* and *communication-based* agility sub-types are viewed as most critical for DSD project success, followed by *methodological*, *time-based*, and *technological agilities*. Finally, *work transition*, *environmental*, and *cultural* agilities were viewed as having the least importance when it comes to enhancing DSD project success.

The first observation that may be made is that, project success requires different types of agility, encompassing resource, process, and linkage-based considerations. This is consistent with recent research that states that a large number of factors are required to complete a project successfully ranging from highly-skilled people to good communication amongst the stakeholders, and the use of a systematic methodology [5].

The fact that people-based agility has the highest salience indicates the importance of having qualified personnel readily available to join the project whenever necessary, along with the ability of project team-members to relocate with ease to different locations, and is in alignment with the consistent argument in prior research, which highlight people to be the most important components of such projects.

The quality of communicative linkage among the important stakeholders is considered a key factor for project success. Our results too reflect this effect. The geographical and temporal distance inherent in the distributed ISD context makes such communicative linkages even more critical, as the resulting communication delays and coordination issues could have serious implications on project success.

The setting up and adhering to appropriate methodologies (i.e. methodological agility) was found to be important, given that they provide structure for effective and flexible utilization of people and technology. In fact, it has been argued that “poor software development [methodological] practices” place significant risk on a software development project [12, p. 190]. Research suggests that demonstrated methodological agility through the tailoring of conventional SDLC phases (e.g. [3]),

or through combinations of agile methodologies [6] may lead to success in DSD projects, and provide validation for the importance of this agility type.

Further, the emergent importance of temporal agility signifies the fact that avoiding delays due to time is a key factor in the minds of individuals involved in DSD projects. Finally, given that our study involved ISD projects accomplished across time and space, where the team focuses on the development of an information system by utilizing different tools, and collaborates using a variety of computer-mediated communication tools, the importance of technology agility is also understandable.

Our results also indicate that work transition-based agility, cultural agility, and environmental agility are not considered to be of great importance. The perceived insignificance of work transition-based agility in our study may be explained by the skepticism surrounding the feasibility of such agility. Balasubramaniam et al. [3, p. 43] comment that “while 24×7 development is sometimes claimed to be a benefit of distributed development, this was far from reality...”. Further, we also feel that the lack of importance of cultural agility may be due to the fact that the extended exposure to different cultures, with the maturity of the offshoring model, has made this agility almost a “given” in most teams, thereby reducing its perceived importance towards project success. Finally, environmental agility pertains to a capability of distributed teams to prepare, and react to catastrophic upheavals in the distributed team’s environment. However, such catastrophic events are rare, and may explain to some extent why our respondents did not consider such agility particularly important. We feel that our research provides important directions to practitioners by providing guidance about possible agility dimensions to focus on. However at the same time we feel that it would be important to conclude with some thoughts on how such agility may be achieved. We draw on the empirical research conducted by Sarker and Sarker [14] to provide some tactical advice on how organizations may strive to achieve the agility dimensions found important in our study. These tactics are summarized in Table 7.4.

7.6 Conclusions

While there is a growing recognition that the adoption of agility principles can enhance the success of DSD projects, there is a limited empirical understanding regarding the effect of agility on project success. This study contributes by empirically examining practitioners’ views surrounding the key criteria for evaluating DSD project success and relative importance of different sub-types of agility to DSD project success. Our study indicates that on-time delivery and effective collaboration form important factors that lead to project success. Further our study indicates that for overall agility to be achieved, an organization needs to focus on all three broad categories of agility-resource based, process-based and linkage-based. We also find evidence that each of these categories of agility may be achieved by focusing on particular sub-dimensions of each of these categories. Specifically our results indicate that resource based agility may be achieved by focusing primarily on people-based

Table 7.4 Tactics for enhancing agility dimensions

Agility Type	Tactic
People-based agility	<ul style="list-style-type: none"> ● Select appropriate locations for overseas operations based on availability of skilled resources, enabling rapid ramping up/down of team ● Maintain strong skilled resources at different vendor locations ● Increase interchangeability of roles, reconfigurability of team, distributed decision making across the distributed teams <ul style="list-style-type: none"> ○ Increase team member versatility across locations ○ Increase diversity of team member skills across locations ○ Rotate personnel across locations to disseminate knowledge and increase shared frame of reference
Communication-based agility	<ul style="list-style-type: none"> ● Increase continuous awareness and visibility of distributed team members <ul style="list-style-type: none"> ○ Appoint key individuals in each location to provide each other with daily updates ○ Increase shared electronic space (through blogs, bulletin boards) reduce geographical/temporal distance ● Increase maturity of inter-locational communication <ul style="list-style-type: none"> ○ Increase social connection and gradual increase of trust among team members ○ Rotate resources across location ● Increase close collaboration among clients and distributed team members <ul style="list-style-type: none"> ○ Create open line of communication with team members ○ Assigning selected members to close physical proximity with clients ○ Promote client visit and interaction to distributed team locations
Methodology-based agility	<ul style="list-style-type: none"> ● Increase the use of prototyping which includes remote team members in addition to team members in proximity to clients ● Tailor conventional SDLC to reduce its rigidity <ul style="list-style-type: none"> ○ Shorter time periods for phases ○ Iterative releases with shorter time periods ○ Increasing capability of carrying out any phase at any location ○ Including “agile” perspectives in an overall waterfall framework ● Appropriate harnessing of Agile methodologies <ul style="list-style-type: none"> ○ Non-dogmatic use ○ Tailoring principles to suit context ○ Identifying aspects of methodology that are more appropriate and discarding less appropriate aspects (see [6]). ○ Disseminating and increasing consciousness about benefits of agile methodologies
Time-based agility	<ul style="list-style-type: none"> ● Promote seamless transition of work across time-zones <ul style="list-style-type: none"> ○ Create uniform work practices and similar norms for interaction across locations ○ Appoint right personnel/roles in time zones consistent with flow of work ● Increase synchronous meeting times, <ul style="list-style-type: none"> ○ Make sure that such meetings are not biased against the physiological and social clocks of particular location ○ Use periodic “swap times” in distributed location shifts to match time zones
Technology-based agility	<ul style="list-style-type: none"> ● Bypass local public infrastructure ● Invest in globally standardized technology enabling locational transparency ● Make available a portfolio of communication technologies ● Create centralized development and tracking environment

agility and to a lesser extent on technology-based agility, process-based agility may be achieved by focusing on methodology-based agility and finally linkage-based agility maybe achieved by focusing primarily on communication-based agility and to lesser extent on time-based agility. We also provide some actionable tips that we hope would provide guidance to organizations in achieving these relatively important sub-dimensions of agility.

References

1. Baccarini, D. (1999). The logical framework for defining project success. *Project Management Journal*, 30(4), 25–32.
2. Bagchi, P., & Rao, R. P. (1992). Decision-making in mergers: An application of the analytic hierarchy process. *Managerial and Decision Economics*, 13, 91–99.
3. Balasubramaniam, R., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile? *Communications of the ACM*, 49(10), 41–46.
4. Borjesson, A., & Mathiassen, L. (2005). Improving software organizations: Agility challenges and implications. *Information Technology & People*, 18(4), 359–382.
5. Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *The Journal of Systems and Software*, 81, 961–971.
6. Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices. *European Journal of Information Systems*, 15(2), 200–213.
7. Holmqvist, M., & Pessi, K. (2006). Agility through scenario development and continuous implementation: A global aftermarket logistics case. *European Journal of Information Systems*, 15(2), 146–158.
8. Jiang, J. J., & Klein, G. (1999). Risks to different aspects of system success. *Information and Management*, 36, 263–272.
9. Kotlarsky, J., & Oshri, I. (2005). Social ties, knowledge sharing and successful collaboration in globally distributed system development projects. *European Journal of Information Systems*, 14(1), 37–28.
10. Lee, O. K., Banerjee, P., Lim, K. H., Kumar, K., van Hillegersberg, J., & Wei, K. K. (2006). Aligning IT components to achieve agility in globally distributed system development. *Communications of the ACM*, 49(10), 49–54.
11. Lui, T. W., & Piccoli, G. (2007). Degrees of agility: Implications for information system design and firm strategy. In K. C. DeSouza (Ed.), *Agile information systems: Conceptualization, construction, and management* (pp. 122–133). Burlington: Butterworth-Heinemann.
12. Procaccino, J. D., Verner, J. M., Darter, M. E., & Amadio, W. J. (2005). Toward predicting software development success from the perspective of practitioners: An exploratory Bayesian model. *Journal of Information Technology*, 20(3), 187–200.
13. Saaty, T. L. (1994). How to make a decision: The analytic hierarchy process. *Interfaces*, 24(6), 19–43.
14. Sarker, S., & Sarker, S. (2009). Exploring agility in distributed information systems development teams: An interpretive study in an offshoring context. *Information Systems Research*, 20(3), 440–461.
15. Sarker, S., Munson, C. L., Sarker, S., & Chakraborty, S. (2009). Assessing relative contributions of the facets of agility to distributed systems development success: An analytic hierarchy process approach. *European Journal of Information Systems*, 18(4), 285–299.

Chapter 8

Preparing your Offshore Organization for Agility: Experiences in India

Jayakanth Srinivasan

Abstract Two strategies that have significantly changed the way we conventionally think about managing software development and sustainment are the family of development approaches collectively referred to as agile methods, and the distribution of development efforts on a global scale. When you combine the two strategies, organizations have to address not only the technical challenges that arise from introducing new ways of working, but more importantly have to manage the ‘soft’ factors that if ignored lead to hard challenges. Using two case studies of distributed agile software development in India we illustrate the areas that organizations need to be aware of when transitioning work to India. The key issues that we emphasize are the need to recruit and retain personnel; the importance of teaching, mentoring and coaching; the need to manage customer expectations; the criticality of well-articulated senior leadership vision and commitment; and the reality of operating in a heterogeneous process environment.

8.1 Introduction

Economic forces are said to be relentlessly turning national markets into global markets, and spawning new forms of competition and cooperation that reach across national boundaries [1]. Thus the importance of reduced cycle time for developing and marketing products (both within emerging markets and globally), and availability of human capital emerge as key drivers to distribute work across geographies. When companies attempt to maximize the benefits of globally distributing work and leveraging available capabilities, the resultant working environment becomes multi-site and multi-cultural. Companies now have to address not only the technical

J. Srinivasan
Mälardalen University, Västerås, Sweden

J. Srinivasan (✉)
Massachusetts Institute of Technology, Cambridge, USA
e-mail: jksrini@mit.edu

issues associated with a new way of planning and executing software projects, but also the associated social and cultural challenges.

In parallel to the changes introduced by distributed development, the family of software development approaches that constitute ‘agile methods’, have forced an equally important paradigm shift [2] in the mechanics and management of software development. We use the word paradigm shift because these techniques have resulted in what is essentially an epochal change. The four values and twelve principles espoused in the agile manifesto that was published in 2001 [3], challenge the conventional notions of a mechanistic model of software development, and focus on a more organic, human-centric approach. The last eight years have seen increased adoption by practitioners on the use of these methods, and reported benefits in terms of increased employee morale [4], productivity [5] and customer satisfaction [6].

In the ideal case, companies would like to be able to combine the strengths of these emerging techniques, while mitigating their weaknesses. In this chapter, we discuss how companies can prepare their offshore organizations for agility using case studies of two companies that have offshored a part of their operations to India called AgileCo and BankCo respectively.

The rest of this chapter is organized as follows: we first set the stage for distributed agile software development in India, followed by a discussion of the individual cases separately, and then synthesize the lessons learned from the case studies.

8.2 Distributed Agile Software Development in India

The main reasons for seeking a services partner in India are flexible resources, skills available, cost savings, and professionalism [7]. Unfortunately the initial ‘over the wall’ approach to software development, in which requirements were developed at the customer site and then handed over to the business analysts in India to flesh out and implement, failed to demonstrate success. Thus, transitioning to an agile approach promised a greater success. The four key barriers in implementing agility in India as pointed by Summers are [7]: culture, communication, working practices, and single vision.

Martin Fowler [8] wrote about the experience of running offshore projects in India and Australia, wherein he emphasized the importance of retaining their high standards in selecting employees (offering jobs to 1 in 200 applicants), maintaining a mix of new hires and seasoned employees, and mentoring developers in the new office on the use of agile methods. Of the 11 lessons learned, two are worth mentioning here: *don’t underestimate the culture change* and *expect to need more documentation*. One of the fundamental challenges of transitioning to an agile environment is the increased autonomy that it provides to the software development team, and the resultant flattening of the decision hierarchy. Thus the importance of creating an agile infected culture, while noting the uniqueness of the local culture has been emphasized [9].

The journey towards adopting agility has already started in India. Wipro’s three year journey in adopting agile methods began to proactively understand, focus and

deliver to customer needs [10]. Their experiences in 90 projects, mentored through a central team of certified scrum masters, which has enabled them to create a model to transition from collocated agile teams to distributed agile teams. The eight best practices they have identified can be understood from two different perspectives: a technical perspective (partitioning stories based on functionality and story-wise collocation of teams; using design structure matrices to understand functional dependencies; technical scrums to prevent speculative coding; visual controls on all sites to create shared understanding and promote collective ownership), and cultural aspects (rotation of personnel between on-shore and offshore to distribute business knowledge; creating a shared understanding between customers and offshore teams through site visits; dedicated video conference rooms; photo chart of the entire team). Similarly, Shrinivasvadhvani and Panicker [11] emphasize the need for mentoring for successful agile adoption. As Batra notes [12], the principles and values provide an effective framework for thinking about the challenges of distributed agile development. However, we take a different standpoint that it is not impossible to build high-performance teams in a distributed/outsourced/off-shored environment. In fact implementing agile methods successfully enhances the competence of the team as a whole, and leads to growing expertise within the organization. While we agree with Cusumano's assessment [13] that truly distributed agile development will require new contracting mechanisms and enhanced communication and coordination approaches, effective distributed agile development is at its core about finding alignment between the values and principles of agile manifesto and the values and practices of the organization adopting it. The referenced reports provide a set of best practices that organizations can leverage to apply agile methods in distributed development.

8.3 Experiences from AgileCo

8.3.1 Case Overview

AgileCo was started in 2001 as the Indian arm of a global software services provider. Given that their parent company had a long history of applying agile development, AgileCo also adopted agile methods, but faced significant challenges in terms of educating their personnel in the use of agile methods and in managing stakeholder expectations. In 2005, at the time of the study, AgileCo consisted of about 75 people and was expected to double in size by 2006. We conducted 12 interviews at AgileCo in two sessions spanning a week in total. Our interviewees included three senior managers, four business analysts, and five developers. Additionally, we observed the interactions of members of AgileCo both intra-organizational and inter-organizational, when we were teaching them during two workshops. This was supplemented with notes from interactions during breaks between interviews, and after-work social events.

Over the 2001–2009 timeframe AgileCo has become one of the benchmarks of agile adoption and usage in India. In addition to mentoring other organizations in the

Table 8.1 Company 1 overview

AgileCo	
Number of developers	75
When was agile introduced	2001
Domain	Software Services
Agile practices used in the organization	Pair-Programming, Planning Game, Daily Standup, 40 hr work week, Collective Ownership

adoption of agile methods, their staff was encouraged to share their understanding of agile methods through conference papers, teaching tutorials, and participation in local knowledge networks. We focus the discussion of AgileCo along the key aspects of *personnel selection and training*, *teaching/mentoring*, and *managing customer expectations*. These aspects emerged as the key components that made the adoption of agility at AgileCo sustainable and successful.

8.3.2 Personnel Selection and Training

AgileCo was highly selective in picking people for their organization. In addition to technical excellence, they focused on ability to adapt to the organizational culture as well as growth potential within the organization. As one senior manager noted:

The selection process we have put into place makes sure that we don't get any duds—we look at coding and aptitude, and at least two interviews. As far as growth within the organization goes, intelligence, communication and technical skills are the foundation, how fast you grow however, is a function of attitude and willingness to find benefits for the customer.

Personnel selection at AgileCo can be broadly divided into three categories, new hires (straight out of school), experienced technical personnel, and experienced managers. The details of the hiring process for each of these classes are different as the role expectations are different; however, the overarching structure is similar to the process shown in Fig. 8.1.

New hires are selected straight from school through both a college recruitment program, as well as through open calls. In addition to assessing their analytical capabilities, the foundational technical skills such as data structures, algorithms, and basic programming are judged. To account for a significant variation in process understanding, every one of their hires straight out of college is put through a rigorous

**Fig. 8.1** AgileCo recruitment process

training program that teaches them the standard processes that are used within the organization. In addition to classroom lectures, the new recruits work in teams to create software solutions to problems that have already been solved for actual customers by the organization. This exposes them to both the mechanics of the process, as well as gives them increased technical skills. Given that India forms their largest recruitment center for people straight out of school, AgileCo's parent company has centered their training program for all of their new recruits in India. As their director for training noted:

We expect to do a lot more work in India, so it's good for the young people recruited outside India to come experience Indian culture, learn the process from expert teachers, and build their own learning in a 'safe environment'.

On a lighter note, it was mentioned by multiple people that the training program initially was referred to as a 'boot camp', but that created issues with respect to getting visas for the global new hires, leading to a renaming of the training program to 'AgileCo University'. When talking to some of the global new hires that were present at the time of the study, one of them pointed:

The great part about coming to India, besides all the great training is that we can get hand-made suits that are tailored—we cannot get that at this price point back home.

Practical Tip: It is important to create a common underlying culture through extensive training within the organization, albeit with local cultural variations. The investment is significant, but the rewards are proportionally greater because it enables individuals to experience the unique cultures of all the nationalities within the organization; enables leaders within the organization to be teachers; creates a common base of work practices that individuals can then expand from; and emphasizes the importance and relevance of the local operations.

A candidate that has successfully completed initial steps and reached the HR interview is further assessed for the cultural fit in the organization. AgileCo's flat organization structure and focus on open and honest dialogue requires the willingness of team members to be active listeners, and to offer constructive critique to ideas that are offered. Additionally, their use of first names in the office, irrespective of organizational rank, can be difficult to people coming from a very hierarchical organizational background. These attitudes and values that are necessary for building an agile organization.

Another strategy that AgileCo effectively employs to understand the fit of senior personnel to the organization, is to use junior members as part of the interview team. This provides the job candidates with an exposure to AgileCo's culture, and provides the team with an opportunity to assess whether or not he/she would be able to work effectively in their environment. When it comes to hiring managers, AgileCo's emphasis is on assessing the individual's ability to mentor, motivate, and manage their teams. Given that a significant portion of their technical team is under

the age of 25, it becomes critical that managers have a deep understanding of the processes used by AgileCo, and can effectively coach their team members.

Practical Tip: Given India's premier position in the software services market, the available talent base is large but the true skill set of a given individual often varies from the picture presented by their resume. AgileCo's experience suggests that a rigorous recruitment process should pay attention to both technical competence and cultural fit.

8.3.3 Teaching and Mentoring

Although AgileCo was dominated by young professionals with less than three years of experience (in 2005) similar to many other Indian fast growing companies, they blended these young professionals with experienced team members, some of whom had not used agile methods prior to joining AgileCo. The advantage of having these experienced people is their deep belief in the process that AgileCo follows, with an ability to articulate from previous experience as to why the process worked. As a senior developer noted:

People often miss the rigor of agile methods. One of the things that I have found over my career is that implementing CMM is commercially unviable—A lot of people that went through the CMM experience with me, became the best XP programmers. One of the things with agile methods, is that if you are in the organization for more than two or three years, is that you have to be really good at what you do or you are extraordinarily lucky!

Practical Tip: The majority of experienced people in the Indian job market have been trained on plan-based development approaches within organizations with high process maturity (as assessed using the CMM or CMMI framework). The experience in AgileCo shows that when experienced people are able to combine the discipline of generating the requisite process artifacts at the appropriate level of detail, with the technical rigor associated with agile software development, they act as the change agents that drive successful agile adoption.

The experienced personnel also act as mentors to younger personnel. When discussing mentoring, the expression that was used often was that AgileCo was a *vil-lage without doors*, and you are really not considered to be a senior member until people ask for your help. When discussing the role of experienced personnel in the organization, a senior manager noted:

You have a responsibility to make sure that junior members of the team get face time with the customer and have the ability to actually deliver a solution. You have to set up the iterations such that they can see how the customer uses the system, and make them gain

a perspective that even when the solution seems sub-optimal from a feature standpoint, it probably delivers greater value from a solution standpoint to the end customer. We started an on-site project with six experienced people and three freshers (junior members)—midway through the project, we lost four of our experienced people, but the customers still valued the contributions of the team.

While the mentor-mentee relationship is both formally and informally enforced within AgileCo, another role that experienced people perform is peer-to-peer mentoring. All of the experienced members operate in a relatively ego-free environment, and recognize the critical role that it plays in AgileCo's culture. As one of the managers noted:

When another senior colleague came on board, he wanted to put his stamp on the way things are done—we as an organization understand that, but he also has to understand the importance of what the organization currently has. It would not be proper to call our relationship a mentor-mentee relationship. Rather it is a peer-to-peer influence that I get to exert that has allowed him to adapt to the organization culture.

What their experienced team members have been able to do, is foster a culture of open communication and highlight the importance of taking responsibility for one's actions. They emphasize the importance of individuality within the organization's value framework. A case in point was the story of a young mentee who had incorrectly escalated the issue to senior leadership, and received a highly cryptic response. His mentor was wondering how a mentee would respond to the situation, and whether he should intervene to support him. As we were discussing the situation, the mentor received an email from the mentee, stating that he had made a mistake and he had called the people involved, and had listed a set of actions he was taking to fix the mess.

Practical Tip: A large number of people in the Indian job market are either used to a hierarchical governance structure with controlled customer access (if they come from plan-based organizations), or are inexperienced and have to be shielded from the customer. The switch to a more fluid development approach requires adaptability on the part of the individual as well as transparency on the part of the team. Mentoring of people either by their peers or by their more experienced managers is critical to effective agile adoption.

8.3.4 Managing Customer Expectations

In 2005, AgileCo was faced with having a customer base that was not well versed in agile terminology. A case in point was pair-programming, wherein the customer questioned the idea of having two people programming together on a single system. They felt that they should not be paying for two people when only one was doing the work. AgileCo came up with the notion of ideal hours—the amount of time it would take to solve a given problem and chose to bill customers that way. One of

the tensions of doing that was that the development team perceived that it was a comparison across the number of hours someone worked on a feature as opposed to the quality of the work. As the manager involved noted:

One of things that I learned to do was to protect the team from the customer and executive management. We told the client that we were going to ideal hours, and instead of being adversarial with my team, I told them to give me ammunition to sell their ideas. Over a couple of iterations, I developed a conversion factor that effectively translated a story point to a standard number of hours. By keeping the story boards updated prior to a standup, I had all the information that I needed for the leadership calls, without having to gather ‘overhead’ data from my team.

Practical Tip: Novel work practices require innovative metrics. In the case of AgileCo, it was recognizing the need to create a common vocabulary with the customer while retaining team effectiveness. The ‘story points to ideal hours’ conversion is illustrative of creating metrics without fundamentally changing a functional measurement system.

As AgileCo continues to grow, it has to ensure that it does not become too inflexible to tailor the process to the problem. Their team-based culture, ability to mix personnel with varied experience levels, and ability to maximize the strengths of their geographical location makes them viable in the long run. As one of the business analysts visiting AgileCo from the parent office noted:

This is the purest form of agile in the entire company. Anywhere else, I have to operate under the pressures of my client, but here, I am shielded from that, and can execute the process while, at the same time, delivering great customer value.

8.4 Experience from BankCo

8.4.1 Case Overview

BankCo is a new organization that was formed in early 2005 in India to support the internal software needs of their global parent organization, which specializes in the banking sector. While some maintenance operations had been previously offshored to a team in India prior to the formation of BankCo, agile methods were not used by the legacy teams, and they had not developed any new products.

Our interviews included a vice president of the parent organization, the CEO of BankCo, their agile coach, a senior project manager, and three developers. One of the developers was a contractor with little agile experience, the second had recently joined after two years at AgileCo, and the third developer had joined after a short stay at another organization that had been using agile methods as their primary development strategy. Complementing AgileCo experience, observations at BankCo suggest that the impact of the senior leadership vision is critical, along with the ability to mitigate the challenges of working in a heterogeneous process environment and implementing proper agile coaching.

Table 8.2 Company 2 overview

BankCo	
Number of developers	20
When was agile introduced	2005
Domain	Banking
Agile practices used in the organization	Planning Game, Daily Standup, Sprint Planning, Storyboards as Information Radiators

8.4.2 Impact of Senior Leadership Vision

BankCo's CEO had a clear vision for where he wanted the organization to be. His vision for growing the business was articulated as follows:

We will do the exact opposite of building a software factory—we will hire talented people who are equal to their counterparts in London and Paris—we will build quality products using talented people, and grow the business by moving projects onshore and taking larger chunks of existing projects—there will be a natural convergence of projects ending up here.

This view was echoed by the VP of their parent organization, who talked about the cost differentials of operating in India, and the larger talent base that was available in India. The current approach of revenue generation by BankCo is to charge the parent organization located in Europe on a unit-cost-per-person basis, such that they can invest in local projects and still make the books balance. When asked about the expected size of the organization, BankCo CEO said:

I expect to have about 100 people here, anything more will be a bonus and any less and the attrition rates would be too high.

Their intent in adopting agile methods was to rapidly demonstrate value to their parent organization, while gaining maximum utility from the capabilities of their talent base.

Practical Tip: Having senior leadership understand the dynamics of using agile methods allows them to articulate a realizable vision, and provides an anchor point for assessing progress towards achieving that vision.

Reflecting the vision set forth by their CEO, BankCo has been extremely selective in who they recruited into the organization. While BankCo is looking to hire across a broad spectrum of experience levels, their recruitment strategy is seen to focus on hiring people who:

- have not peaked in their careers,
- and have the ability to think independently.

Given their need to grow to meet their size requirements, they have used a strategy of hiring contractors to take on junior developmental roles while their permanent

staff takes on the senior roles of program managers, business analysts and senior developmental roles. As one of the contract employees noted:

This is so different from my own organization which is CMMI level 5, I have more autonomy here than with my own organization, and we (BankCo) actually get better quality products delivered faster. In my organization, we have a system that is based on tenure—people get promoted just because they had been in the organization long enough—here it is purely a meritocracy.

In addition to looking for strong talent, BankCo also focused on ‘fit’ with the corporate culture. BankCo recognizes that their team has to operate in close conjunction to their European counterparts—being able to communicate their expectations is extremely important. Given that their parent company’s culture is to manage by exception, BankCo’s CEO has placed a significant emphasis on employee empowerment and open bi-directional communication across the entire organization. The emphasis on recruiting people that can spot and deliver value has led to rejecting a large number of job applicants.

Practical Tip: A large number of companies starting their operation in India leverage the capabilities of consultants drawn from other organizations. By clearly defining roles and expectations of consultants and employees, BankCo has enabled a smooth culture transition.

8.4.3 Heterogeneous Process Environment

BankCo currently has green-field teams that use agile methods as their core development methodology as well as teams involved in sustainment that operate using a traditional plan-driven approach. Although one would expect a culture clash between the agile and non-agile teams, they have successfully avoided the problem by having the same project manager for both sets of teams. This project manager served as the bridge between the teams, infusing the open communications and flattening the hierarchical structure in the plan-based development teams, and bringing in greater documentation discipline to the agile teams. As the project manager noted:

When I first got here, it was as if these teams existed in separate worlds—it didn’t help that the team doing agile work was physically located in a different building than the rest of the teams, but the issue was more of product support versus product ownership. Now we have greater interactions across these teams, and are migrating best practices between them.

Practical Tip: Having a common project manager working across two different development approaches provides a means of creating a shared understanding between the teams, and driving towards a common baseline process.

8.4.4 Agile Coaching

One of the greatest strengths of the agile team was the presence of an agile coach to train the team in the use of agile methods. As one developer noted:

Working with the agile coach was brilliant—brilliant—brilliant—when I came to BankCo earlier this year, he was very different from anyone I had ever met before—he was calm professional and very passionate about work. We were unaware of agile, refactoring, test driven development. He sat with each of us, showed us how to do things. When we didn't have a wall for the story board, he came up with the idea of using the mobile white board that you see here.

From a mentoring standpoint, the CEO noted that with the exception of the coach, he had really not put anything else in place. His expectation was that the teams themselves would self organize and that the senior members of the team would serve as mentors to the less experienced members, and that the open bi-directional communication, coupled with a flat organization structure would enable peer-to-peer interactions as well.

When talking about the impact of the coach on mentoring, the contract developer noted:

I am actually getting feedback about myself for the first time in my career (this comes from a person with 8 years of experience)—the right feedback that was required, and more importantly, he made me think about what other directions I needed to think about. He encouraged the process of thinking as opposed to telling me what to do. The feedback was not always positive, but it was put in a constructive manner. He also shared a lot of stories about his experiences when his projects were cancelled, how he felt, the possible risks of the project we were currently doing—little things on how to make myself better.

Practical Tip: An agile coach with experience in working in a global market is critical to successful adoption and sustainment of agile methods.

BankCo's teams have both technical and business competencies. Their CEO emphasized the need for every team member to a clear understanding of where they fit in the organization and how their role would evolve. This was fundamental both to growth as well to prevent long-term attrition. In 2005, BankCo was still in the early stages of its agile adoption. The clear vision articulated by their CEO, coupled with strong project management and coaching has considerably eased their transition to using agile methods. Their emphasis on mentoring and adherence to the agile principles within their development approaches has enabled them to incorporate the philosophy more easily into their organization. As their CEO noted, they are currently in the middle of a change, and only time will tell if the change was successful.

8.5 Conclusions

In synthesizing the learning from the two cases, we highlight the importance of senior leadership vision in adopting and sustaining agile methods. In the case of

BankCo, the driver for adopting agile methods was to show the increased value provided by the new offshored organization. They were able to effectively translate the vision provided by their CEO to their work practices, as seen through the vision articulated by their coach.

Organizations often deal with some degree of heterogeneity in work practices. In the case of BankCo, it was addressed by using a unique project management structure with the same project manager dealing with both sets of teams. In the case of AgileCo, it was addressed by having the project manager serve as the translator between the India-based team and their international customer. It is important to point out that heterogeneity is in and of itself, not detrimental to the organization. The challenge is in finding the right level of standardization that does not inhibit team autonomy, and at the same time provides sufficient predictability to support senior leadership decision-making.

Both organizations have emphasized the importance of creating and maintaining an agile-infused culture. As one of the senior members at AgileCo noted:

We have achieved success through the nature of the projects we work on, and in indoctrinating our junior members.

AgileCo's success in using agile methods as an organization-wide standard has enabled them to indoctrinate less experienced people in the process, and yet, balance out that indoctrination through the use of more experienced personnel, most of whom came from plan-based development environments. The intuitive notion that such senior personnel would find it hard to function in an agile environment, not embrace it and act as impediments to change was proved wrong. These mentors maintained an oral history of the limitations of plan-based approaches, but at the same time, emphasized the discipline it takes to execute agile processes. Without the presence of experienced personnel, organizations run the risk of 'by-the-book' agile implementations that at a surface-level reflect the adoption of agile practices, but do not result either in the organization level transformation or the increased delivery quality that is expected.

One of the limitations of indoctrination is the literal and rigid interpretation of agile methods that are espoused by junior members. This was one of the issues that came up over dinner conversations when someone talking about a more structured process like the Rational Unified Process was dismissed by the entire table, with none of the junior members at the table being able to make a coherent argument for why RUP would not work in their organizational context.

At the fundamental level, the adoption of agile approaches has to change the nature of the work associated with software development, while at the same time fostering deeper understanding of the system being developed. From our perspective, agile approaches focus on providing a means of doing the work of software development and can drive grass-roots level transformation. Whereas CMMI and other top-down process improvement efforts provide the means of creating policy-driven change. That is not to say that organizations can just morph into following agile methods. There needs to be significant discipline on the part of senior leadership to support the process, and shield the teams utilizing these processes from pressures from the external environment. For instance, the project manager has the

responsibility of preventing the team from thrashing either due to customer variation or due to resource challenges. Since using agile methods drives the organization towards the use of high performance team structures, new incentive and performance management systems need to be developed.

In summary, the key success factors to successful implementation of agile development in the studied Indian companies were:

- Designing a human capital strategy that supports growing an agile-infected culture
- Creating a shared language to communicate within and across organizational boundaries
- Finding a balance between experienced and inexperienced personnel to ensure effective project management
- Establishing formal and informal organizational learning mechanisms
- Mentoring to institutionalize work practices
- Crafting incentives to increase adoption of agile practices

Acknowledgements The research presented in this chapter was supported in part by the Lean Advancement Initiative at MIT, and the Swedish Foundation for Strategic Research through the PROGRESS Center at Mälardalen University. Preliminary versions of this chapter were improved based on comments from Prof. Kristina Lundqvist, Prof. Christer Norström and Dr. Gustav Naeser, and two anonymous reviewers. Also, the chapter benefitted significantly from the editorial feedback received from Dr. Darja Smite.

References

1. Herbsleb, J., & Moitra, D. (2001). Global software development. *IEEE Software*, 18(2), 16–20.
2. Kuhn, T. (1970). *The structure of scientific revolutions. 1962*. Chicago: University of Chicago Press.
3. Beck, K. et al. (2001). *The agile manifesto*. The Agile Alliance.
4. Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131–133.
5. Cohn, M., & Ford, D. (2003). Introducing an agile process to an organization. *Computer*, 36(6), 74–78.
6. Williams, L., & Cockburn, A. (2003). Agile software development: It's about feedback and change. *Computer*, 39–43.
7. Summers, M. (2008). Insights into an agile adventure with offshore partners. In *Agile 2008*.
8. Fowler, M. (2007). Using an agile software process with offshore development. Electronic. Available: <http://www.martinfowler.com/articles/agileOffshore.html>, p. 12-07.
9. Doshi, C., & Doshi, D. (2009). A peek into an agile infected culture. In *Agile 2009*. New York: IEEE.
10. Sureshchandra, K., & Shrinivasavadhani, J. (2008). Adopting agile in distributed development. In *IEEE international conference on global software engineering*.

11. Shrinivasavadhani, J., & Panicker, V. (2008). Remote mentoring a distributed agile team. In *Agile 2008*.
12. Batra, D. (2009). Modified agile practices for outsourced software projects. *Communications of the ACM*, 52(9), 143–148.
13. Cusumano, M. (2008). Managing software development in globally distributed teams. *Communications of the ACM*, 51(2), 15–17.

Part III

Management

Chapter 9

Improving Global Development Using Agile

How Agile Processes Can Improve Productivity in Large Distributed Projects

Alberto Avritzer, Francois Bronsard,
and Gilberto Matos

Abstract Global development promises important productivity and capability advantages over centralized work by optimally allocating tasks according to locality, expertise or cost. All too often, global development also introduces a different set of communication and coordination challenges that can negate all the expected benefits and even cause project failures. Most common problems have to do with building trust or quick feedback loops between distributed teams, or with the integration of globally developed components. Agile processes tend to emphasize the intensity of communication, and would seem to be negatively impacted by team distribution. In our experience, these challenges can be overcome, and agile processes can address some of the pitfalls of global development more effectively than plan-driven development. This chapter discusses how to address the difficulties faced when adapting agile processes to global development and the improvements to global development that adopting agile can produce.

9.1 Introduction

Globally distributed software development projects present special challenges for agile processes since the primary tools that agile processes use to effectively solve complex problems rely on frequent communication and quick feedback [5, 9]. Distribution directly affects these capabilities by introducing delays and limiting the bandwidth and quality of communication. Thus, agile processes must be specifically customized to preserve their qualities in the face of the communication constraints

A. Avritzer (✉) · F. Bronsard · G. Matos
Siemens Corporate Research, 755 College rd East, Princeton, NJ 08540, USA
e-mail: alberto.avritzer@siemens.com

F. Bronsard
e-mail: francois.bronsard@siemens.com

G. Matos
e-mail: gilberto.matos@siemens.com

of distributed environments. Once such customization is done, however, agile processes can effectively address many of the challenges faced by large distributed projects [8] and result in extremely productive projects [3, 10].

From our experience on large and small projects within Siemens AG, we have seen first-hand many of the challenges of using agile in distributed projects and the many strategies that were tried and adopted to address them. Some of the challenges for agile are well-known (e.g., the difficulty of communications across distant time-zones) but others have not been discussed previously (e.g. supporting a formal requirements engineering process, while doing agile development). Furthermore, we have also witnessed how, once these challenges were addressed, the agile processes led to improved productivity in contrast to waterfall or plan driven processes.

This chapter is organized as follows: We start with a short description of the projects in which we participated and present specific success factors and challenges of these projects. We follow by discussing how global projects can adapt agile processes and which issues they must consider. In the next section, we discuss those characteristics of agile processes that are critical to address the challenges of global projects and how to preserve them when adapting agile. Finally, we conclude with a short summary of how to apply agile processes in large distributed projects.

9.2 The Projects

The discussion presented here is strongly inspired by two large distributed projects in which the authors were key participants. The first project lasted for three years and involved approximately one hundred developers divided into ten teams distributed across the USA, Europe and India. The goal of the project was to build a platform for a product line of PC-based control and monitoring systems to replace a collection of legacy systems that supported different lines of business. The main challenges faced by this project were the adoption of multiple new technologies and the design constraints of making a generic platform.

This global agile project faced significant risks. First, there were communication challenges: for example, between the teams farthest apart (USA and India) there was only one overlapping working hour. Because of the expected length of the project, it was not practical to ask the teams to shift their workday to increase the overlap. The most serious risk, however, was that the project relied on multiple untested or very immature technologies. The project adopted agile processes, in part, because of their ability to handle volatile technologies.

The project enjoyed a number of advantages. First, the development teams included a large number of developers that had previously worked on the development or the maintenance of the legacy products that the project aimed to replace. Thus, most teams could count on local domain expertise to quickly resolve or clarify domain questions. Second, the testing process could use sample workflow and test cases based on the usage of these legacy products. Finally, most members of the project had previously participated in global projects and many key members had lived abroad and were accustomed to the cultural differences between the sites. As

a result, although cultural differences are frequently cited as a significant risk for distributed projects, they were not a factor in this project.

Table 9.1 Project overview

The First Project	
Number of Developers	≈ 100
Number of Teams	10
Duration	3 years
Status	Launched, ongoing incremental releases
Agile practices	Scrum
Involved locations	USA-5, Europe 3, India 2

The second project was divided into two large subprojects: one working on a software platform for distributed control of independent embedded devices and the second using that platform to provide a distributed administration and control capability. The platform subproject was initiated first and was developed mainly by co-located teams in Germany, while the application subproject started with several distributed teams seeded from the platform project teams and working with the evolving platform. Both projects were considered a significant success that resulted in multiple releases of applications with market differentiating features over four years and continued management support.

Table 9.2 Project Overview

Platform_and_Application	
Number of Developers	100
Number of Teams	15
Duration	4 years
Status	Launched, ongoing incremental releases
Agile practices	XP, Scrum
Involved locations	Germany-4 locations, US 2, Greece 1

Our experience is based on the application subproject, which faced a more distributed environment, more volatile requirements and technology dependencies, and much more critical deadline visibility due to the planned end-user deployment. A large risk was that the project had to rely heavily on remote subject matter experts for clarifying domain specific issues. The dependency on the evolving platform project introduced an ongoing technological risk.

The most important factors in the success of this project were the existence of an initial team with significant expertise in the targeted development technologies, and the faithful implementation of agile principles that allowed the development team to effectively tackle the largest risks during the first 2 iterations.

Our experience is also drawn from a number of smaller agile distributed projects with five to twenty developers, lasting for several months to a year. In these projects we were involved in development roles and in agility mentor roles [6]. To the extent that these projects faced issues similar to those of the larger projects and addressed them in the same way, they provide further validation for the value of agile techniques on distributed projects.

9.3 Deploying Agile Techniques in Global Projects

This section examines the issues to address when adopting agile in a global project. We discuss the factors to consider when selecting or adapting the various agile techniques, and illustrate the possible solutions with examples from our projects. We first discuss some simple organizational issues to set the stage for the discussion of the communication issues and the process issues. Then we discuss some project planning issues and various tooling and technical issues.

9.3.1 *Organizational Issues*

The first issue to confront when adopting an agile distributed process is how to organize the teams. There are two models of organization:

Co-located Teams In this model, the teams are isolated across geographies and integrated by a Scrum of Scrums that meets regularly across geographies. In our projects, most teams were organized as co-located teams and we relied heavily on the Scrum of Scrums meetings to coordinate the inter-team communications as discussed below.

Distributed Teams In this model, Scrum teams are cross-functional with members distributed across different locations. The main argument in favor of this approach is that if there is good communication between the team members and if they all have high expertise, the resulting team exhibits very high productivity [10]. Our projects included some distributed teams with varying degrees of success. Some successful distributed teams were initially co-located teams for an extended period of time, in our project examples, five to six months. After that period they became distributed and continued to perform well.

Pairing of Developers We have found that many project features tended to be cross-disciplinary, and that it is a good practice to have the development done by two to three developers working together, therefore providing some of the benefits of pair programming. In our experience, pairing of developers worked best when people were grouped because they had complementary skills (e.g., a tester and two developers, or an architect, a UI designer, and a developer), or for experienced developers mentoring less experienced ones.

Travel and Delegation No matter the teams' organization, it is important to occasionally get the teams together. If possible, co-locating the teams for the first one or two iterations will build trust and greatly help future communication. We used the travel and delegation approach in one project and developers found it helpful. Afterwards, having some of the team's members change locations occasionally for short periods of time, or getting many developers together temporarily, e.g., to tackle some complex issue, helped maintain that trust and team spirit.

As a special case of this last example, we found that for infrastructure or architectural change with project-wide implications or to introduce a new technology, it was very useful to have such change tackled first by a co-located team made of representatives from all the teams affected by the change. With such an approach, all the implications of that change can be detected and evaluated quickly and by co-locating this *change* team we allow them to learn together, hence more quickly, the intricacies of the new infrastructure, architecture or technology. Afterward, the participants can serve as mentors in their home teams. For example, in one project, we used an application framework that saw three upgrades during the project. For the first two upgrades, we relied on the co-located change team approach and could integrate the upgrades in two to three weeks. For the third upgrade, we used non-collocated teams and ended up encountering upgrade problems across many teams for four to five weeks.

9.3.2 *Communication Issues*

The foremost issues to address when using agile in global projects are the communication issues. Most agile techniques rely on frequent communications and short feedback cycles, and these present special challenges in a global project. Significant effort needs to be invested up-front to lessen the communication requirements of the project. This means the design of a good architecture that addresses the most important non-functional requirements. It also means accepting occasional code duplication rather than refactoring, since refactoring requires more communication.

It is important to enable the communication required by the agile process rather than trying to control the communication flow through extensive up-front analysis.

Inter-team Communication Our approach to agile was closely inspired by the SCRUM method, hence we relied on the scrum-of-scrums meeting to generate the overall picture of the state of the project and, more importantly, to communicate critical information between teams. In one project, we had a daily Scrum of Scrum meeting and, in the other, two to three weekly ones.

In general, the scrum-of-scrums is a forum for discussing the project wide issues and for identifying team dependencies, which are then communicated to the relevant team members in order to set up direct communication channels. We commonly used the scrum-of-scrums for prioritization and backlog reviews, and for resource planning. The scrum-of-scrums is also the first step in escalating any problem that

can't be resolved at the level of a single team, such as unavailability of external resources for priority tasks, negotiation of interfaces and work split between teams.

The selection of a good representative from each team is critical: the representatives must have enough technical understanding of the project that they can effectively summarize their team's work, understand the summaries of the work of the other teams, and, more importantly, understand the implications for their teams of the work done in the other teams. For example, it's a mistake to think that the Scrum of Scrums should be a Scrum of ScrumMasters. If the ScrumMaster for a team is not a technical member, then it is one of the technical members who should participate in the Scrum of Scrums.

Informal Communication *Informal communications* refers to the informal and spontaneous "corridor talk" that happens between members of a co-located team. Informal communications keep the team members informed of what is happening in the project on a day-to-day basis and is often a critical knowledge sharing mechanism. In our projects, we relied mostly on the *distributed Scrum of Scrums* model of team organization, so the various teams were co-located and could still rely on the informal communication channels. For inter-teams communications, we relied on phone, e-mail, IM, video conference, and web conference to allow people to communicate relatively easily. We had also various information sharing spaces (Wiki sites, SharePoint) for people to share knowledge in a structured and persistent form.

Formal Communication By *formal communications* we mean the more formal, official communications like reporting project status, reporting and escalating issues, issuing milestone reports, etc. Although agile tend to eschew such process documentation, this is usually not an option for large projects. Such projects have to be planned and monitored at least from a financial standpoint. This entails documenting preliminary efforts and cost estimates and performing a full lifecycle planning. Furthermore, large projects will also require reporting on the major milestones achieved by the projects. In our projects, this was addressed by adding corresponding documentation tasks to the backlog of tasks.

Communication with Domain Experts Domain experts are critical, but it is difficult to obtain these resources. This is particularly acute for long projects, since it is usually not possible to spare these resources for the duration of the project. In one project, we were given sets of very high level generic feature descriptions and had no access to the domain experts to validate the actual detailed scenarios of how the artifacts were being created, processed and presented. Since the development team was missing the needed level of expertise in the application domain to independently refine the features, we made several attempts to create prototypes that would minimize the needed verification time by the domain experts. We had to declare the project in crisis during its first iteration before being given the needed access to domain experts. The quick prototyping approaches [7] continued to be useful in minimizing the needed effort for the domain experts and in getting more relevant and wide ranging feedback from them as the development team was building up its experience in the domain issues.

The emphasis of agile processes on creating some deliverables early contributed to some initial friction among the stakeholders, but ensured that the needed domain experts would be made available to the project. The push for including a domain expert in the creation of some early deliverables was a pivotal moment for the project, and once this input became available, the project quickly reached a state of continuous progress with new features and architectural improvements being implemented at close to initial estimates.

9.3.3 Process Issues

Agile processes being non-prescriptive, it is normal that different agile projects exhibit significant process differences. In fact, any agile team should keep evaluating and refining their process to better meet the project needs. This flexibility, however, carries the risk of weakening the process and causing problems with project delivery or quality. When adapting an agile process, one should remember that a key characteristic of many agile techniques is that they induce a *positive feedback loop* where the technique improves productivity in a self-reinforcing way. This is important to keep in mind when process issues come up and the project must decide how to adapt them.

Transition Out of the Legacy Process Managing legacy processes while transitioning to an agile process is a politically delicate matter as support for existing processes is often required during the transition period. We recommend transforming the existing development process into a coarse process with only the major milestones remaining. Therefore, the organization could follow the agile process while continuing to deliver the documents and artifacts required by the traditional process at the major milestones. In our projects, this approach added some documentation tasks but not a significant number.

Process Refinements The development process itself should be treated as a project artifact, thus, it should be defined based on best practices, and maintained through frequent reviews and refactoring. It is best to start with a view on a vertical slice of the process, and refine it. A first broad view should reveal what development specialties will be needed (e.g., build automation, test automation, UI design, architecture, etc.). Once these specialties have been identified, the initial resource can be allocated, and the relevant expert can refine the needs of the process from the standpoint of that specialized need.

During the sprint or during the sprint retrospectives we can review how the process worked and whether it needs tuning. Keeping this process under review ensures that constraints that are more process-oriented in nature (availability of build experts, availability of domain experts, etc. . . .) are taken into account during planning. For example, if the domain experts for the back-end were less available during a sprint, we would concentrate more on front-end features.

In one project, the Scrum of scrums was used to discuss not only the current state of the system but also to discuss and revise the development process. The revision of the development process at the Scrum of scrums meeting helped distribute and promote to the complete group whatever process variations were experimented and found useful by one team. In particular, some of the process discussion that would normally happen in end-of-sprint retrospectives took place during the scrum of scrums meetings. Thus, the process could be updated quickly.

Backlogs It is important to have a well-fed backlog of tasks so that “lag times” can be used productively and to encourage correct prioritization: It is important to make clear to the product owner that the backlog is a tool for them to control the value of the features that are being worked on, and that it is better to have a backlog that is clearly too big than one that misses relevant tasks! Thus, product owner prioritizes features to make sure high value ones are done first, and the low priority or questionable ones are subject to subsequent decisions. This is particularly important for a large project because the long duration of the project makes it likely to face changing needs and priorities of the features.

Minimal Up-front Planning The key planning concern is to enable the initial development and to provide a solid base to validate the most important non-functional requirements. Thus, the analysis must identify the main features of the system to be developed, how they will be used by the most important stakeholders, and the main components and services that will make up the system. The initial iterations are used to validate the architecture, based on the main functional and non-functional features of the system. The preliminary analysis and planning only need to identify and enumerate these features and organize them into relevant vertical slices that can be tackled by the development teams.

Many projects try to enhance agile processes by doing somewhat detailed estimation of the required effort. In all of our projects, the early estimation of complex projects proved to be too optimistic, and caused conflicts related to resource and project planning. The projects that used a more transparent process and a dynamic approach for reprioritizing features were able to detect errors in the estimation of the required effort early in the project lifecycle, and could therefore adjust the level of resources required in time to meet the project delivery times. Subsequent projects/releases with the same teams were able to provide significant incremental updates completely within budget and schedule.

Agile Form Without Function Agile processes are meant to increase transparency in the conduct of complex development and problem solving endeavors. The associated disciplines and practices are tools that should help with this goal, but are not goals in themselves. It happens occasionally that teams adopt the agile process and present their adherence to the practices as a deliverable goal, without really providing a valuable and verifiable output. On a low level, this behavior can be seen when Scrum team members define tasks of indefinable value to the product owner, such as documenting a design or preparing a quality assurance plan, or if the

daily Scrum talks primarily about meetings had and meetings planned. On a higher level, a team iteration review may face a situation where a number of features are completed but can't be demonstrated or verified because these features are marked as "this is just an iteration".

This situation of agile form without agile function is primarily found in recent (or unwilling) converts to agile, and in situations where the developers don't trust and therefore don't want to provide transparency to the product owner or other management stakeholders. While agility clearly strives to empower the developers to address important technical issues even if they are not directly visible to the customer, it is very important to enforce some level of transparency in the work because this is the primary mechanism for building trust and effective communication. It is never required that all tasks from a given iteration could be demonstrated to the product owner, but all agile processes (SCRUM in particular) demand that every iteration review must have valuable output demonstrated and validated. Various supporting tasks should be aggregated into valuable features that could be demonstrated to work, for planning and progress reporting purposes.

9.3.4 Tools and Technical Issues

Although no single tool is strictly mandatory for a successful agile project, the right set of tools greatly facilitate realizing the various agile processes.

Communication Tools Providing good communication tools (e-mail, Instant Messaging, phone conferencing, web conferencing, remote computer sharing, etc...) is a must. We found it a good practice to provide a palette of tools so that people with different styles of interactions can find whichever tool suit them best. Early co-located face-to-face meetings were quite appreciated by the teams and were important to building the required trust among the team members.

We also used knowledge sharing tools like Wikis or SharePoint. These tools were quite useful to provide a shared site to report on the status of distributed efforts (e.g., the testing of a new release), although the information provided there tended to have a very short shelf-life. Some items, though, like architectural samples or code examples were useful over a long period of time.

Continuous Integration In both projects, we found centralized continuous integration to be a critical enabler. Being able to build and run the full system (even if in a partial state) frequently, and to have access to the system from all sites was essential to avoid serious integration problems and to ensure that misunderstandings were quickly revealed and addressed.

In ours, and others projects within our organization, we found that where continuous integration was achieved early, and was prioritized higher than any other feature, the projects achieved steady progress. Conversely, in several other projects, where the progress on individual features was prioritized above the preservation

of the integrated system, there were consistent integration problems. In addition, these integration problems were compounded in global projects by the challenges of distributed communication which led to inconsistent understanding between the teams of the purpose and usage of specific locally developed features. However, in projects where the continuous integration process was achieved at a later point in the lifecycle, noticeable improvement in productivity and progress were observed.

Integration Tools To enable continuous integration, build automation tools are required. In our projects we used CruiseControl to rebuild the project whenever an update was made to the code base. Should that build fails, a notification was sent to all users that had contributed to that update. This process ensured that any build failures were detected in a timely manner. If needed, after a short delay, the technical leads would get involved in trying to identify the error, regardless of which developer introduced it. The primary responsibility of each team was to ensure that their errors do not stay in the repository after their work hours. This generally meant that a few team members looked at every build failure, and in most cases, the build was restored during the same day without impacting the other teams and location. Although, it would have been possible to automatically roll-back the code base to the version of the last successful build, we do not recommend this practice: the ultimate goal of the continuous integration is to push for a resolution of integration problems. Therefore, we want to encourage developers to check in their code and detect integration problems. Finally, for quality control, we used test automation tools and ran them as part of the continuous integration build.

Configurations Management We recommend a single main trunk for the code base as we did in one project. A single main trunk requires extra work to schedule updates from multiple sites, and to prevent bottlenecks on shared files. However, a single main trunk enables true continuous integration. In the other project, we had site or team based branches and the integration team merged all the branches prior to integration. This reduced the interaction overhead but resulted in a stream of serious integration problems and resulted in a reduced availability of the fully integrated application, and contributed to delaying the discovery of incompatibilities.

Centralized configuration repository may be a bottleneck for multiple distributed teams working on the same code base. Remote repositories may impose significant network latency, and slow down the individual developers. We found that having local repositories with a multi-site merging protocol provided good performance with acceptably low instances of conflict. In order to minimize the update conflicts, we marked a limited set of files as being critical for sharing, and allocated specific time-of-day update guidelines to different teams. With this refinement, the distributed access to centralized configuration management reached satisfactory performance. Table 9.3 summarizes our recommendations for adapting Agile for globally distributed software development projects.

Table 9.3 Adapting agile for global projects

Issues	Recommendation	Details
Teams Organization	Co-located teams	Using co-located teams is simpler and they can become distributed teams after a while
	Co-locate early work	Co-located work is a good way of creating an environment with trust and effective communication. It is particularly critical for distributed teams to develop trust and understanding of each others capabilities
	Co-locate critical work	Important or risky architecture changes, whether internally motivated or driven by external dependencies, should be performed in a co-located team, with delegated experts from relevant teams
Inter-teams communication	Scrum of Scrums with technical representatives	The Scrum of scrums must cover both the technical and managerial issues spanning multiple teams but the technical issues must take precedence
Legacy process	Reduce the legacy process to major milestones	Provide the documentation required of the major milestones of the legacy process
Process refinement	Treat process as a project artifact	The process should be updated and refined as needed. It is important however to ensure that the refinements preserve the positive feedback loops of agile
	Use Scrum of scrums to discuss process	Using the Scrum of Scrums to discuss process improvement was more effective than waiting for the sprint retrospective
Integration issues	Continuous integration	We recommend an automated build process with optional regression tests and failure reports
	Single main trunk in CM system	This forces the integration issues to arise, and be fixed, early on

9.4 Improving Global Projects Using Agile Processes

Agile processes were initially applied on smaller projects with good results in terms of cost, quality and customer satisfaction. The practical experience indicates that size and complexity are not an impediment for effectively using agile processes. Our project experience suggests that agile processes are very effective in addressing the needs of large distributed projects as long as incremental improvements can be leveraged into a successful application or system.

Effective Preparation Work A common problem for large projects is to correctly decide how much time and effort should be spent on architecture and high-level design for the project, before starting the development work.

The use of agile forces development teams to start with constructive, deliverable work as soon as possible. The core requirement of agile is that the teams *must*

have demonstrable results by the end of each iteration. To meet the end of the iteration deadline, each team and product owner will have to make decisions on what achievable objectives they want to prioritize in the current iteration. The architectural analysis, design, and modeling tasks are not dropped, but must be addressed in the context of delivering end-user features. This approach has dual advantages over a pre-development architecture phase. First, the experience and feedback on the developed or prototyped features increases the body of knowledge available to make architectural decisions. Second, it becomes easier to determine when the results of preparation work are sufficient and have high enough priority to be assigned to feature development.

In our experience, the more tightly integrated the deliverables are from an early stage of the project, the better the distributed collaboration proceeds. Conversely, the more distributed projects depend on local development, and on a subsequent central integration phase, the more problems projects have in both phases: development and integration. By doing local development without the benefit of an integrated application, the distributed teams sometimes work without access to the understanding of the shared domain, and may have difficulty delivering correct functionality to the integrators.

Product Owner The product owner is expected to provide the developers with a list of desired features, prioritized to represent the interests of the stakeholders. They also need to have the organizational knowledge and support to be able to access to external domain experts. Whenever possible, the product owner should be able to act as an internal domain expert for a majority of features. The product owner should be emphasized as a domain-specific role, rather than being a strictly management role. In large distributed projects, the product owner needs to be fully committed to the project, and to be able to periodically visit the sites and directly interact with the developers in elaborating the features and assessing the progress and planning.

The development teams act as suppliers to the product owner, and needs to keep an open and trusting communication with him/her. In agile projects, this means providing transparent effort estimates of the immediately planned features, generally the ones in the current iteration. It also means that reports about development progress and effort spent need to be frequent and realistic. Finally, the communication should be open, without a scrum master or site manager acting as an intermediary.

Communication—Bandwidth Limited communication bandwidth tends to reduce the quantity and value of communication within a global project, moreover the product owner's and the domain experts' time is usually limited and this further reduces the amount of information that can be communicated. Yet, in spite of its strong reliance on frequent communications, the Agile approach is quite appropriate even in the face of limited bandwidth or difficulties to get the product owners' or the domain experts' time. Agile processes tend to focus on small-scale and short term deliverables and this allows agile teams to effectively communicate on the issues of interest even within the bandwidth limitations of globally distributed projects. Agile

approaches also provide a specific benefit in the area of eliciting knowledge from product owners and domain experts because the ongoing development provides an operational system that can serve as a frame of reference for elaborating incremental features, and for identifying risks and opportunities. By reducing the amount of time and effort that is required from the domain experts, and by focusing on the interesting areas of system interaction, it becomes easier for agile projects to collect the needed input or feedback. It is also more effective to get product owners and domain experts to clarify and resolve issues as they arise, rather than attempting to detect and resolve all potential problems during an idealized planning and designing phase.

Continuous Integration The continuous integration process represents both a supporting technology and a crucial proof of transparency and visibility of development results. The existence of a functioning system base provides the first real verifiable deliverable anchored in reality, and is generally an agile mainstay. Complexity and dependencies in distributed systems often create pressures toward decoupling the work at individual locations, and we have seen a multitude of recurring integration problems in projects with such delayed integration.

The continuous integration process allowed us to do early system and acceptance testing, to make us aware of the possible sources of system instability, and to include mitigation strategies for these instabilities in the planning of the development sprints. Continuous integration also ensures that the various components do not diverge too much, and that everyone has the same understanding of the application. Centralized continuous integration is particularly valuable for distributed projects since it provides these projects with a stable frame of reference for communication, planning, and domain-related feature elaboration.

Distribution of Design and Analysis Work A common approach to the analysis, architecture, and design phases of large distributed projects is to have these phases done in depth by a central team at the beginning of the project. Our experience is contrary! In our projects, and in others (see [1, 2]), we found benefits to a more distributed approach in which multiple development teams get involved at an early stage. This creates open communication channels and develops high level of trust leading to more effective collaboration later on. Agile approaches foster such flat structure and the resulting communication and collaboration channels resulting in better productivity for distributed teams.

Unstable Requirements Agile approaches are particularly effective in dealing with unstable requirements because they provide short feedback cycles to review and re-prioritize features. As the changing competitive landscape and stakeholder interests lead to evolving requirements, most large projects will confront a significant level of requirement volatility simply due to their duration and the low level of initial understanding of what the end-requirements will be. Agile methods allow for the requirement elaboration work to be concentrated on a small set of high priority features, and to incrementally develop a system that incorporates an increasing

number of desired features. By delaying the requirement elaboration work of low priority features to a time that is closer to their implementation, the developers will have better domain and dependency understanding when they start development, and they will also be able to complete the implementation at some acceptable level before significant changes occur.

Introducing or Updating Technologies The emphasis on short iterations and prototyping enables agile methods to efficiently handle new technologies [4]. This still holds in the context of global projects: a new technology can be quickly prototyped in a separate development track as a special feature during an iteration and if the prototype is successful, that technology can be deployed widely in later iterations. Moreover, if the prototype is not successful, then only a limited effort has been expended. The capability to handle new technologies is particularly important for global projects because they tend to last multiple years and, therefore, will operate in an environment of evolving technologies.

Managing Customer Expectations The use of short iterations, frequent feedbacks and a well-managed backlog are powerful techniques to handle customer requirements. The key technique is to keep the backlog very visible and open. With a view of the backlog and frequent prototyping, the customer is able to see how the work is progressing. An important advantage of prototyping is the ability to reposition the project if major deviations from customers' needs are detected. By keeping the backlog open, the customer can add features to the project by inserting them in the backlog and reprioritizing the other features. The referred process gives the product owner the flexibility to control and prioritize the three most important aspects of planning complex development tasks: scope, quality and cost, and to do it in a very fine grained manner, to the level of individual features, based on the needs of the different stakeholders.

Table 9.4 How agile affects global projects

Agile technique	Impact on global projects
Frequent, short iterations with “demoable” results	<ul style="list-style-type: none"> • Avoid excessive up-front planning and design work • Better handling of unstable requirements • Help manage customer expectation
Product owner	<ul style="list-style-type: none"> • Provide a representative for all stakeholders • Keep features prioritized according to changing business reality
Continuous Integration	<ul style="list-style-type: none"> • Avoid late discovery of integration issues • Improve system level understanding and testing in the project
Develop storyboards and prototypes	<ul style="list-style-type: none"> • Storyboards and prototypes are a powerful way to collaborate with remote domain experts on features • Help test new technology before introducing them in projects
Access to domain experts is critical	<ul style="list-style-type: none"> • Access to domain experts is critically important for clarifying both the project goals and specific features or user stories

9.5 Conclusions

Agile development processes have been successfully used in large distributed projects by many companies, and the broadening interest in this area testifies to the success of the methodology. Agile methods address requirement volatility, technology changes and evolving stakeholder interests in a much more effective way than plan-driven development methods. In this aspect, agile methods are well positioned to address the most important types of problems that face engineering organizations in the globally outsourced environment: maintaining global partnerships with best-of-breed service providers in order to achieve fast time-to-market for innovative and high quality products.

Our experiences in a number of global projects using agile support that characterization. We have found that agile can be successfully adapted for globally distributed environments and we have presented here guidelines on how to do such an adaptation. Once a project has successfully adopted agile, the benefits of agile were preserved in globally distributed environments and addressed many of the challenges of such environments.

There are a few key elements to keep in mind when adapting agile for a distributed environment. Simple remote collaboration practices, like providing multiple communication tools and holding regular cross-site meetings, allowed distributed teams to significantly reduce the overhead of distributed collaboration. Co-location for individual teams is important though after a while these teams could become distributed. All teams should be represented during project inception, preferably by co-located delegates. Finally, we found centralized continuous integration to be key to a smooth collaboration of the distributed agile teams.

Once agile processes are adapted to global projects, they effectively address many of the challenges of distribution. They preserve the flexibility needed to deal with volatile requirements and multiple stakeholders. The short iterations and frequent feedbacks clarify the customer needs and give customers an actionable insight into the project's progress. The backlog management provides a very fine grain control that makes it easy to satisfy the customer's customer needs. These observations allow us to conclude that the advantages of agile over plan-driven development are preserved in globally distributed environments.

References

1. Avritzer, A., & Paulish, D. (2010). A comparison of commonly used processes for multi-site software development. In I. Mistrik et al. (Eds.), *Collaborative software engineering*. Berlin: Springer. Chap. 14.
2. Avritzer, A., Hasling, W., & Paulish, D. (2007). Process investigations for the global studio project version 3.0. In *ICGSE 2007. Second IEEE international conference on global software engineering* (pp. 247–251).
3. Berczuk, S. (2007). Back to basics: The role of agile principles in success with a distributed scrum team. In *AGILE '07: Proceedings of the AGILE 2007 conference* (pp. 382–388). Washington: IEEE Computer Society.

4. Cockburn, A., & Highsmith, J. (2001). Agile software development: The business of innovation. *Computer*, 34(9), 120–127.
5. Herbsleb, J. D., & Moitra, D. (2001). Guest editors' introduction: Global software development. *IEEE Software*, 18(2), 16–20.
6. Hwong, B., Matos, G., McKenna, M., Nelson, C., Nikolova, G., Rudorfer, A., Song, X., Tai, G. Y., Tanikella, R., & Wehrwein, B. (2007). Quality improvements from using agile development methods: Lessons learned. In I. G. Stamelos & P. Sfetos (Eds.), *Agile software development quality assurance* (pp. 221–235). Hershey: IGI Publishing.
7. Hwong, B., Matos, G., Rudorfer, A., & Wehrwein, B. (2009). Rapid development techniques. In B. Berenbach, D. Paulish, J. Kazmeier, & A. Rudorfer (Eds.), *Software & systems requirements engineering: In practice* (pp. 233–255). New York: McGraw-Hill, Inc.
8. Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using scrum in distributed agile development: A multiple case study. In *ICGSE 2009. Fourth IEEE international conference on global software engineering* (pp. 195–204).
9. Ramesh, B., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile? *Communications of the ACM*, 49(10), 41–46.
10. Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *System sciences, 2007. HICSS 2007* (pp. 274a–274a). 40th annual Hawaii international conference on software systems. <http://www.scrumalliance.org/resources/17>.

Chapter 10

Turning Time from Enemy into an Ally Using the Pomodoro Technique

Xiaofeng Wang, Federico Gobbo,
and Michael Lane

Abstract Time is one of the most important factors dominating agile software development processes in distributed settings. Effective time management helps agile teams to plan and monitor the work to be performed, and create and maintain a fast yet sustainable pace. The Pomodoro Technique is one promising time management technique. Its application and adaptation in Sourcesense Milan Team surfaced various benefits, challenges and implications for distributed agile software development. Lessons learnt from the experiences of Sourcesense Milan Team can be useful for other distributed agile teams to turn time from enemy into an ally.

10.1 Introduction

Time is a priceless and scarce resource for software development projects [4]. It is especially true in agile software development. A brief review of the 12 agile principles behind the Agile Manifesto reveals that time is an important dimension of agile processes, symbolized by terms such as “early”, “frequently”, “couple of weeks”, “daily”, “regular intervals” in these principles [2]. Agile teams work with short time-boxed iterations and need to maintain a fast yet sustainable pace throughout the project lifespan [3]. When moving to a distributed setting, the time dimension

X. Wang (✉)

Lero, The Irish Software Engineering Research Centre, Limerick, Ireland
e-mail: xiaofeng.wang@lero.ie

F. Gobbo

DICOM Dipartimento di Informatica e Comunicazione, University of Insubria, Via Mazzini 5,
21100 Varese, Italy
e-mail: federico.gobbo@uninsubria.it

M. Lane

Computer Science and Information Systems, University of Limerick, Limerick, Ireland
e-mail: michael.lane@ul.ie

Fig. 10.1 A tomato-shaped timer



is further complicated by issues such as time zones [6], geographical distance [7], and different cultures [1]. However, there is very little reported evidence of effective time management techniques applied in agile software development, especially in the context of distributed teams.

The Pomodoro Technique is a time management tool that was originally intended to optimize personal work and study. More recently, it has been widely applied by Italian agile teams [9]. Awareness of this technique is growing among the wider, international agile community (two tutorials on the Pomodoro Technique have been given in Agile 2009—the international conference). The technique is named after the usage of a common kitchen timer in the shape of a tomato (“pomodoro” in Italian, see Fig. 10.1). The heart of the Pomodoro Technique is 25 minutes of focused, uninterrupted work on one task, then 5 minutes of rest. There are also rules to keep the integrity of pomodoro, and tactics to deal with internal and external interruptions. However, starting as a personal time management tool, how is it applied by an agile team, especially when the team is working in a distributed environment? There is no ready answer in spite of the increasing popularity of the Pomodoro Technique in the agile community.

Based on this observation, the objective of our study is to provide a better understanding of the application of the Pomodoro Technique in agile teams, especially when they work in distributed contexts. To this end, we studied in-depth one agile team that has applied the Pomodoro Technique extensively. The team collaborates with other remote sites of the company where the Pomodoro Technique is not used. This allows us to reflect on the impact of the Pomodoro Technique (and the lack of it) in a distributed context.

The remaining part of the chapter is organized as follows. In the next section we review a set of time-related issues and argue the importance of time management in software development in general and agile software development in a distributed context in particular. It is followed by an introduction of the Pomodoro Technique. Then the experience of Sourcesense Milan Team using the Pomodoro Technique is presented. We analyse their experience and provide useful guidelines for implementing the Pomodoro Technique in the following section. The chapter ends with a conclusion section that highlights the contribution of our study and points out future studies.

10.2 Time Is an Enemy?

Time occupies a crucial place in software development projects. It is one of the most important factors dominating software development processes [10]. And just like many people have experienced, to various degrees, the anxiety associated with the passage of time, many software development projects have suffered from a set of time-related issues. Back in 1975, Brooks claimed that “more software projects have gone awry for lack of calendar time than for all other causes combined” [4]. In [10] a set of interlinked time-related problems in software development processes are independent, including: *bottlenecks*, which occur when one or more functions in the development process are dependent upon the output of another function within the process, resulting in developers having nothing to work on in the meantime; *schedule problems*, both construction of a feasible project schedule and to meet the schedule that has been set; *difficulty in time estimation* of large module/class/task; *time pressure*, which happens typically towards the end of a development process when the development team cannot meet the project schedule either because of poor time estimations or bottlenecks; and *late delivery*, which occurs as a result of inappropriate project planning, usually due to poor estimations.

These inter-related problems, in essence, are all subjects of time management, one of the major knowledge areas of project management [12]. This area involves decomposition of project work into manageable tasks, estimation of task durations, scheduling of tasks, and controlling and monitoring the execution of tasks. Effective time management is crucial for addressing the time-related problems and leading to the success of software development projects. However, [10] argue that in early software engineering projects, when waterfall versions started to emerge, time issues were essentially neglected. As we proceed along the software engineering timeline, time issues receive increasingly more attention and their importance in software development processes is increasingly recognized, as evidenced in Team Software Process (TSP) [11], Rapid Application Development (RAD), and recently in agile software development.

Time plays a more crucial role in agile software development than in conventional waterfall-like software development processes. This is demonstrated by a review of the 12 agile principles [2] from a time perspective. A review of these principles shows that 50% of them have an emphasis on time:

- Our highest priority is to satisfy the customer through **early** and continuous delivery of valuable software.
- Welcome changing requirements, even **late** in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software **frequently**, from **a couple of weeks to a couple of months**, with a preference to the **shorter timescale**.
- Business people and developers must work together **daily** throughout the project.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain **a constant pace** indefinitely.
- **At regular intervals**, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These principles pose new time-related challenges to agile teams, apart from the previously discussed time issues. Agile teams need to work at a fast yet sustainable pace. The risk in having sole focus on velocity of development is the reduction of enthusiasm among team members. This may then have an impact on the sustainability of an agile team's daily work. It is important to achieve this equilibrium in agile software development, but it is underestimated in practice [9]. As a consequence, time management in agile software development not only means effective planning and monitoring of the work to be performed, but also should help agile teams to create and maintain a fast yet sustainable pace.

These two aspects of time management in agile software development can be further complicated as software development moves into global, distributed settings, which is an approach adopted by many current software projects. Issues such as time zones [6], geographical distance [7], and different cultures [13] will affect the previously discussed time-related issues and the effectiveness of time management techniques employed by agile teams. To better understand the impacts of distributed context, it is useful to consider different distributed team configurations (see Grinter et al. [8] for different kinds of team configurations). Each different team structure presents different benefits to the work being undertaken, and would impact the time management technique used by agile teams in different ways.

In spite of the importance of time and time management in agile software development, however, there is a paucity of both time management techniques and studies on the application of these techniques in agile software development, let alone in a distributed agile context. The Pomodoro Technique is one promising time management technique and is increasingly popular in the agile community. A growing number of agile teams use the pomodoro technique within their agile development processes [9].

10.3 The Pomodoro Technique

The goal of the Pomodoro Technique is to encourage consciousness, concentration, and clarity of thought through effective time management. A 'pomodoro' is 25 minutes of focused, uninterrupted work on one task then 5 minutes of complete rest. The inventor claims that, based on scientific proof, "20- to 45-minute time intervals can maximize our attention and mental activity, if followed by a short break" [5]. The 5-minute break aims to support team members in establishing and maintaining an optimal attention curve while engaged in project activities. In order to increase the impact of this effect, following every four consecutive pomodoros a longer pause of 15 minutes is recommended.

The technique can improve the productivity of an individual. Improvements in productivity are achieved through increased motivation and the technique has also proved effective in supporting the management of complex situations. These benefits can be achieved through the following two inter-related aspects of the Pomodoro Technique: *time-boxing* and *duration estimations*.

10.3.1 Pomodoro as Time-box

One of the primary inspirations behind the Pomodoro Technique is time-boxing [5]. Time-boxing suggests that, once a series of activities has been assigned to a given-time interval, the delivery date for these activities should never change. If necessary, the unfinished activities can be reassigned to the following time interval. Corresponding to the idea of time-boxing, to maximize participant concentration, every time-box (or pomodoro), needs to be protected. “Protecting pomodoro” leads to fewer interruptions. There are two kinds of interruptions that need to be addressed:

- *Internal*: These interruptions are triggered by the participant, e.g. “I should check email”, or “I need to get a cup of coffee”;
- *External*: Triggered by other entities, e.g. a phone call or a request from a colleague.

In order to handle these interruptions effectively, an “indivisible rule” needs to be enacted. A pomodoro represents twenty-five minutes of pure work that cannot be split up. There is no such a thing as a half or a quarter of a pomodoro. If a pomodoro is interrupted definitively, i.e. the interruption is not deflected, then the pomodoro is considered to have never commenced—it is made void.

Used as a time-boxing tool, the Pomodoro Technique can help enhance focus and concentration on work by cutting down on interruptions. Consequently, it can alleviate anxiety linked to the passage of time and reduce both time-wasting and overtime. A sustainable working pace can be obtained through the alternation of work and rest and the combination of short breaks and long pauses.

10.3.2 Pomodoro as Unit of Effort

To master and improve the use of the Pomodoro Technique, an underlying daily process is suggested, which consists of five stages:

- *Planning*: to decide the activities to do in the day;
- *Tracking*: to gather raw data on the effort expended and other metrics of interest;
- *Recording*: to compile an archive of daily observations;
- *Processing*: to transform raw data into information; and
- *Visualizing*: to present the information in a format that facilitates understanding and clarifies paths to improvement.

In each stage, the pomodoro plays the role of unit of estimation. Two rules apply: (1) if a task lasts more than 5–7 pomodoros, break it down. Complex activities should be divided into several activities; and (2) if it lasts less than one pomodoro, add it up. Simple tasks can be combined.

Used as an effort estimation tool, the Pomodoro Technique can support the refinement of an effort estimation process through the use of continuous reflection of

team activities (more detailed instructions of how to apply the Pomodoro Technique as a personal time management technique can be found in [5]).

The Pomodoro Technique was invented initially for individual work, as a personal time management tool. But it has been developed and refined in the context of teamwork by the inventor and advocates of the technique over the time. [9] report the technique as a *team* time management tool used by several XP teams. They claim that the Pomodoro Technique is an unstressful—as well as efficient—way to help teams find their “natural” rhythm in daily work. Their study is a good starting point and provides a broad picture for investigating how the Pomodoro Technique can be applied in agile teams. Our study intends to go into more depth to understand the benefits, issues and concerns of using the Pomodoro Technique in an agile team within a distributed context.

10.4 The Application of the Pomodoro Technique in Sourcesense Milan Team

In this section, we present one case study of the application of the Pomodoro Technique in an agile software development team. More specifically, we investigate how Sourcesense Milan Team, an XP team, applied the Pomodoro Technique as both a time-boxing tool and an estimation tool. We also present their reactions to working in a distributed setting, collaborating with other locations that did not use the Pomodoro Technique.

10.4.1 Background of Sourcesense Milan Team

The company, Sourcesense, is a European systems integrator providing consultancy, support and services around key Open Source technologies (Table 10.1).

It is distributed across several countries, as illustrated in Fig. 10.2.

In Italy, there are two teams that are focused primarily on software development: Sourcesense Milan Team and Sourcesense Rome Team. Sourcesense UK contains a team that plays the role of customer proxy: Sourcesense UK Team. The remaining part is based in the Netherlands, Sourcesense Netherlands, and it is responsible for the provision of consultant-oriented services. In the context of this case study, Sourcesense Netherlands is not engaged in any of the projects performed. Table 10.2

Table 10.1 Company overview

Company: Sourcesense	
Number of developers	<50
When was agile introduced	2007
Domain	Open Source Systems integrator

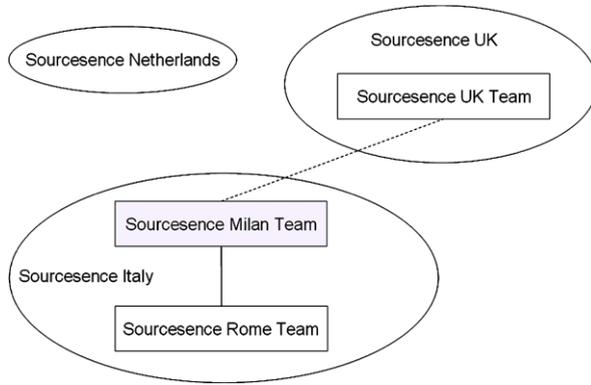


Fig. 10.2 The distribution context of Sourcesense Milan Team

Table 10.2 Team overview

Locations	Number of members	Roles
Sourcesense Milan	10	Developers, XP coach
Sourcesense Rome	2	Developers
Sourcesense UK	1	Customer proxy

is an overview of the distribution context that Sourcesense Milan Team is embedded in. In terms of configurations of distributed teams, in the context of this case study, two structures are in place:

- *Modular*: Sourcesense Milan Team and Sourcesense Rome Team take responsibility for separate modules or features of the system under development.
- *Functional expertise*: Sourcesense UK Team is deemed to be the most appropriate source for requirements elicitation in the absence of customer access.

10.4.2 The Development Process of Sourcesense Milan Team

Sourcesense Milan Team works at a fast pace, restricting itself to 1-week iterations. An iteration planning meeting is held where the work in the current iteration is planned out. User stories are selected for implementation in the iteration and the relevant story cards are posted on the white board in the team’s office. The team also uses online spreadsheets to track the progress of user stories. Because the team does not have an on-site customer, a progress report to the customer detailing all achievements/issues of the previous iteration is compiled and sent out to the customer after the iteration planning meeting.

Generally every two weeks the team conducts a retrospective where the members reflect on their development process. Each team member is given an opportunity to lead retrospectives, not just the coach.

Every Wednesday is set aside for research activities—the team uses their time to focus on the study of both project-related concepts and growth of general work-related skills.

A standup meeting is conducted at 9:30 everyday, generally lasting for no longer than 15 minutes. The first activity performed by the team members is to individually review the previous day’s journal. This is a document produced by the team members at the end of each day outlining the activities performed during the day. It serves multiple purposes: providing non-located team members an immediate view of overall team progress, enabling the team members to achieve closure on their day’s work and acting as a review that is used to set the tone for the following day’s work. Journal reviews are completed before 9:30 each morning as this is the time scheduled for the daily standup meeting. Having used the journal to review what was achieved on the previous day, the two main questions addressed in the standup meeting are: (1) “What am I going to do today?” and (2) “What are the problems in doing it?”

After the standup meeting, the team goes into development mode. The developers work in pairs all the time. Pair rotation happens regularly.

The development process of Sourcesense Milan Team would have been similar to many other XP or agile teams but for their use of the Pomodoro Technique. This is seen more clearly by reviewing the perspectives of Sourcesense Milan Team in relation to the impact of their use of the Pomodoro Technique. Two main areas are presented:

- Having the pomodoro play two key roles in the development approach: “time-box” and “unit of effort”.
- Issues arising from collaborations with distributed teams that do not use the Pomodoro Technique.

10.4.3 Pomodoro as Time-box

Pomodoro is used to time-box the development activities of the team. Several aspects of using pomodoro as a time-boxing tool in the team are highlighted below.

10.4.3.1 One Pomodoro Rules Them All

The initial application of the Pomodoro Technique involved a pomodoro timer for every pair of developers. The owner of the story card was responsible for loading the timer, and updating the card. As a result of the responsibility assumed by each partner within a pair to support their colleague, internal interruptions were minimized.

The coach was responsible for deflecting external interruptions—he protected the integrity of each pomodoro as the pair worked on its activities.

As a result of retrospectives on how best to effectively apply the Pomodoro Technique in a team setting, it was decided to experiment with an extension of the Pomodoro Technique. The name given to the extended approach was “shared pomodoro”. This approach proposed that a pomodoro be applied to the whole team, i.e., “one pomodoro rules them all” (Fig. 10.3 is an illustration of a team working with a shared pomodoro timer).

Although some developers expressed reservations on this approach initially, it is now accepted by the team as a good way to address certain issues that were having an adverse impact on the team. These issues were related to interruptions that were being caused between pairs of developers. When each pair had their own pomodoro and their own associated pomodoro timer, there were cases where they disturbed each other due to different break times. Some pairs preferred to work for 50 minutes and take a longer pause thereafter; others used pomodoro just to track their work without paying attention to the breaks. This resulted in noise and distractions for pairs that were not at break. The shared pomodoro approach ensures that when a pomodoro is being tackled, everybody is working; no one has distractions from other people having a pause. It is also good for the whole team to have long pauses all together. Syncing up the pomodoros can also increase cross-pair communication after breaks. Alignment of breaks enables the team to switch pairs more frequently. When pomodoros were not aligned, it was not possible to switch partners within pairs and there was an increased risk of a pair attempting too much in one session. The shared pomodoro approach also helps the team members to be more disciplined in their adherence to both working time and break time. This helps the team to obtain a working rhythm. The team actually behaves like a team, and the cohesion between the team members is increased. An additional benefit to the shared pomodoro approach is that having breaks shared with other pairs acts as a motivational device. This is especially true for pairs that find themselves less than enthusiastic about returning to work due to the types of tasks that are required to be done in their pomodoros at a given time. Not everybody can get to work on the “cool” tasks all the time—but having the opportunity to share in the overall environment at break time helps to overcome this issue.

The team also realizes that shared pomodoro should not be used to block potentially constructive interactions during a 25-minute time slot. They believe that the Pomodoro Technique and shared pomodoro in particular, is intended to make the team members more aware and respectful of everyone’s work. Interruptions that should be avoided include phone calls, instant messages, and people wandering in and requesting assistance from the team members on non-project-related activities. However, as stated above, there are acceptable interruptions. The team works in a face-to-face setting in an open office. The coach observes that if he hears someone having trouble he would intervene and help them. He believes that it might be worthwhile to interrupt his work for a minute to save the other pair from maybe 30 minutes of “*puzzling over something that I know straight away*”.

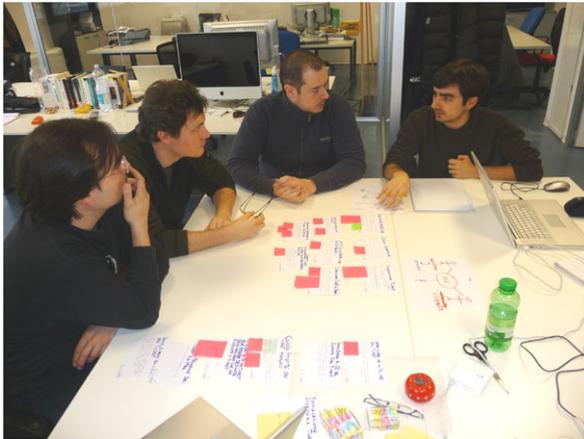


Fig. 10.3 Team engaged in planning using the shared pomodoro approach

10.4.3.2 Break is Break

The team realizes that the 5-minute break is as important as the 25-minute working time for them. The team is well aware of the fact that breaks may be used in a variety of ways. The uses of a 5-minute break can be viewed as a continuum, ranging from complete relaxation (day dreaming, practicing Qigong, sleeping), to more active breaks, such as responding to emails, reading blogs, or even discussing what is just been completed in the last pomodoro with colleagues. According to the team, it takes training to rest effectively, just like it takes training to be able to develop software. In accordance with the recommendations of the Pomodoro Technique the team treats the breaks seriously and the developers are not encouraged to do activities during the break. For example, the team keeps a personal machine separate from work machines. This computer is situated in a different location than the work machines. The intention is to discourage team members from checking emails and reading blogs during the breaks. However, it should be noted that the team members do check emails from time to time on their laptops. The coach admits that, even though it is a sign of indiscipline, it does happen occasionally.

10.4.3.3 Time-boxing Non-development Activities

Pomodoro is used by the team to time-box not only development, but also other activities, such as study and meetings.

As previously stated, Wednesday is scheduled for team study. The team spends four pomodoros studying various topics on Wednesday afternoons. To the team, study is not a break—customer-focus is maintained during these activities. One of the goals of study activities is to promote lateral thinking—“*to try to solve the problem from different angles*”. The team needs to get quick feedback on what is learnt.

Therefore, stories used to drive study activities should be as small as possible in order to shorten feedback cycles. Time-boxing study time with the Pomodoro Technique could help. Another application of this technique is meetings. These events are also time-boxed with pomodoro, in order to help people focus and reduce wasted time.

10.4.4 Pomodoro as a Unit of Effort

Pomodoro is used extensively, as a unit of effort, in planning, estimating and tracking progress. In the case of Sourcesense Milan Team, the unit of effort is one pomodoro per pair. For example, there are 6 full-time people in the team (3 pairs). If each pair works on 10 pomodoros per day, the total team capacity is 30 pomodoros per day.

Activities that do not require team members to work in pairs, such as administrative meetings, are measured by half-pomodoro per person. For example, if a meeting takes 5 developers half an hour, the total effort the team spends on the meeting counts for 2.5 pomodoros.

10.4.4.1 Pomodoros vs. Abstract Story Points

The team originally estimated user stories in abstract story points, which is a common practice suggested by XP [3]. A story point is an abstract complexity measure of a user story. Initially, the team did not commit to the estimation of durations for each story point. In time, the team switched to directly estimating everything in pomodoro per pair. To the team, story points started losing meaning. Instead, the pomodoro became a more concrete measure of effort.

Another concept utilized by the team, following the suggestion of the original Pomodoro Technique inventor, was “pomodoro type”. This concept classifies pomodoros spent into different categories, such as analyzing, coding, refactoring and testing, to help the team to obtain a better understanding of the effort spent on different types of work, e.g., the ratio of coding over refactoring, or how much effort spent on “*wrestling with the server*” in deployment.

10.4.4.2 Tracking Pomodoros

Every morning the team members pick up the story cards from the whiteboard (as shown in Fig. 10.4) and the current story cards that are on the desk of the developers. Usually each card holder marks a cross on the back of the story card for each pomodoro consumed. This helps to track how much real effort was spent on a story. Every evening they will put all the cards together again on the whiteboard, to understand how many pomodoros of user stories are left unfinished, and if they can finish

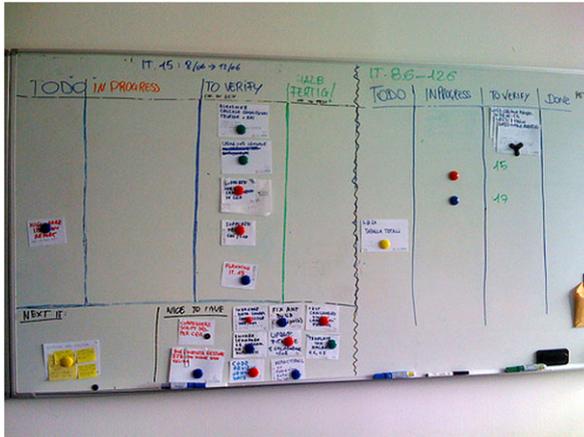


Fig. 10.4 The whiteboard with user stories in current iteration

them within the iteration. It also helps to measure how much uninterrupted work the team can do in a day. They have a shared spreadsheet to store historical data about each iteration. The white board only shows the current iteration.

10.4.4.3 When Not to Measure or Track with Pomodoro

The team realizes that certain activities do not require estimation and also may not require that the effort spent on them be tracked. The typical activity that the team does not track is non-project-related exploration. There are two types of exploration: project related, called spikes, and non project related, more exploratory study that is driven by the interests of the team members rather than by project issues. Spikes, which are related to any user story, are tracked using the Pomodoro Technique. In contrast, the team believes that general exploration, which is described as “*useful exercises of dreaming where you would like to be or to do*”, should be conducted without time pressure. The team often conducts experiments collectively on the code base without the objective of production work. They believe that the result of exploration should be shared among the team members. Therefore, presentations of new technology and new ideas are held from time to time. There is also the concept of a “lunch cinema club”. The team members get together and watch videos from important agile authors on the big TV. When time permits, these viewing sessions are then followed by a short discussion. All these activities are not tracked in pomodoros.

10.4.5 Addressing Remote Collaboration with Teams That Do Not Employ the Pomodoro Technique

Sourcesense Milan Team operates in a larger, distributed environment, as previously described. However, the Pomodoro Technique has not been adopted by the other sites. Sourcesense Milan Team members feel that the use of the Pomodoro Technique to pace and time-box each day “*protects*” them from interruptions, because they believe that “*two or three chats open, reading emails while working—you can’t say it’s concentrated work*”. The application of the Pomodoro Technique also enhances the teamness of Sourcesense Milan Team. The team members feel that they are working as a real team, where internal interruptions are minimized and the coach can protect the team from external interruptions. They also feel that their application of the Pomodoro Technique makes the project work more transparent. They are able to demonstrate to other sites how many hours of the day are spent on development, which helps to build the trust of other sites on Sourcesense Milan Team.

The application of the Pomodoro Technique in Sourcesense Milan Team created a team working style which could be different from the other sites where the Pomodoro Technique is not used, which in turn may enlarge the mismatch of team cultures at different sites. For example, although Sourcesense Rome Team is located within the same national borders as Sourcesense Milan Team and functions as a development site as well, the site is felt to be “*just as distant as the Netherlands*”. The fact that the Rome Team does not apply the Pomodoro Technique may have an adverse influence on this perceived distance.

The other distributed sites (Netherlands and UK) are more engaged in consulting activities rather than software development. The nature of the work decides that many of the staff there works as individuals rather than in teams. In such a context, it is more difficult for them to be free from all sorts of interruptions. They are not as “protected” as Sourcesense Milan Team by the Pomodoro Technique.

However, Sourcesense Milan Team is aware of the existence of different working styles and team cultures at the different sites, and believes that they should not impose the Pomodoro Technique on the other sites. They are concerned that their use of the Pomodoro Technique may act as a barrier to effective collaboration, as the coach put it sincerely:

The difficulty is in communication, being reliable, and trying to make yourself useful. You have the risk of presenting a wall like ‘this is our method, we do it in this way, and you have to work around us’. This is not what the method is meant to be. So you have to learn to be humble enough to be helpful to your colleagues, not just with your customers.

10.5 Turning Time into an Ally

In this section we reflect on the experiences of Sourcesense Milan Team using the Pomodoro Technique, and draw some practical implications.

10.5.1 Shared Pomodoro

The shared pomodoro concept used in Sourcesense Milan Team is an adaptation of the Pomodoro Technique that was invented as a personal time management tool originally. The team has obtained positive results using shared pomodoro. However, although not specifically highlighted in the case study, it should be noted that there are several potential drawbacks to the use of this technique. The working rhythm of the team is liable to be interrupted by any team member's necessary and unavoidable pause during a pomodoro. It takes more effort to coordinate the whole team to start the first pomodoro when people come in to the office at different times in the morning. Similarly, commencing a new pomodoro after short or long breaks may be delayed. The whole team has to wait for all the team members coming back to the open space. It is possible that one pair may be working on a really challenging issue, and following completion of the pomodoro they may need a slightly longer break than others. Conversely, another pair may work on a relatively easy task and wish to take a shorter break. These different physical and psychological impacts are not easily reconciled by shared pomodoro.

Practical Tip: To understand whether shared pomodoro is right for a team, the team needs to know how confident each individual team member is with the rhythm set by using the basic Pomodoro Technique. If they are already using it in a disciplined manner and are comfortable in the environment, then the team can try shared pomodoro for 4–6 weeks. At that point, they may then decide whether or not to adopt this extension of the technique. Consultation with the team about adoption of this approach is preferable to managerial imposition of it as a mandatory process. Otherwise, there is a risk of making the developers uncooperative, because they could feel their freedom and creativity are being constrained by a shared timer.

The experience of Sourcesense Milan Team also reveals that the Pomodoro Technique should be used to block internal or external interruptions, not constructive interactions among team members. Communication and interaction are a key tenet of any agile method. The Pomodoro Technique helps a team to be aware of positive interactions and supports the deflection of unnecessary and unconstructive interactions/interruptions. However, it is up to the team to use the technique in a sensible way to balance maximum concentration of the team and effective interactions among the team members.

10.5.2 Collective Breaks

Breaks are taken seriously by Sourcesense Milan Team and the team members are encouraged to relax and recuperate during their break. Using shared pomodoro can

help the team members to take breaks regularly, but does not guarantee that team members take proper breaks. The very fact that the team members are taking breaks altogether would tend to make breaks more like the extension of a pomodoro. A simple chat could easily slip into a technical discussion of the work just completed in the previous pomodoro. Such a situation could result in the break being just as taxing as the work itself. The case of Sourcesense Milan Team presents some good practices to foster relaxing breaks, such as placing the personal machine far from the working machine.

Practical Tip: In Sourcesense Milan Team, the team members also take breaks individually. This does not specifically promote relaxation. The shared pomodoro concept can be extended to breaks as well. A collective break can be taken where team members do some relaxing activity together during breaks, such as game playing, 5-minute fitness club, etc. Collective breaks can help to avoid the situation where a pair of developers does not wish to let go of their work following the signal from the pomodoro timer.

10.5.3 Estimation and Tracking

Sourcesense Milan Team uses a pomodoro per pair as the unit of effort in estimation and tracking. Pomodoro is actual time measure in contrast to abstract story points, or “gummy bears”. Consider the purpose of estimation and tracking in agile software development. Estimation of user stories is more about granularity of tasks than about effort needed to implement them, and tracking effort spent on them is less about amount of work done than about learning and improving the team’s capability of estimating throughout. Using actual time makes estimation and tracking more transparent and learning more concrete.

However, estimates and tracking results should not be used as measurement of each individual team member’s performance, otherwise there will be a tendency to game the number, and the Pomodoro Technique used as a Tayloristic approach to exploit the team or a micro management tool for project management.

10.5.4 One Pomodoro Rules All Sites?

In Sourcesense Milan Team, the Pomodoro Technique is not used in a truly distributed manner. But the technique is not confined to co-located teams and can be adapted and used in distributed settings and in a distributed manner. Just as the expansion of the Pomodoro Technique from a personal time management tool to a team management tool surfaced various complexities, various benefits and challenges are likely to emerge when the technique is applied in a distributed context.

Practical Tip: Remote pairs or distributed team members could apply the shared pomodoro approach as a time-pacing tool to synchronize and coordinate their activities. It can help better manage interruptions that are generally an indispensable element of distributed development, such as emails, instant messages and phone calls. The shared pomodoro approach can be implemented in distributed teams using virtual space and virtual timer technologies (a virtual timer on a server), whereas pauses can be shared via Skype or analogue systems.

If distributed team members keep their individual pomodoro timers, it would be more difficult for them to be aware if others are in the middle of a pomodoro or not, since the working status is not as obvious as in a co-located team where it can be understood at a glance. To address this issue, the working status needs to be made more explicit and visible to each of the distributed members. For example, *cherrytomato* (<http://www.chrylers.com/cherrytomato/>), a software tool that intends to support the distributed pomodoro technique, integrates a virtual pomodoro timer with instant messaging tools such as Skype. When a developer starts a new pomodoro, his status in Skype would be switched to “do not disturb” automatically and can inform other team members how many minutes are left for the current pomodoro or show other customized messages.

The experiences of Sourcesense Milan Team also indicate that, if used properly, the Pomodoro Technique can increase the transparency of both estimates and project-related work expended at different sites. Consequently, this may help to build up trust among different sites. There are a growing number of agile planning tools that support distributed estimation and tracking. The Pomodoro Technique can be easily integrated in these tools to support estimating and tracking in pomodoros.

However, the issues that are associated with shared pomodoro in a co-located team may become more challenging when temporal, geographical and social/culture distances are involved. For example, it will take greater effort to coordinate the whole distributed team to start the first shared pomodoro. In instances where the team is distributed across different time zones, developers will be involved in different phases of a working day, resulting in possible variances in concentration levels between team members. Therefore, the balance of maximum concentration of team members and effective interactions among them may be more difficult to maintain in a distributed context. Last but not least it needs to be cautioned that the Pomodoro Technique should not be used as a Tayloristic approach to exploit outsourced sites.

10.6 Conclusions

In this chapter we presented a case study of the application of the Pomodoro Technique in an agile software development team. The effects of the technique on the

team were analysed through two angles: pomodoro as timebox and pomodoro as unit of effort. We argued the benefits and potential issues of using one pomodoro for the whole team, team breaks, estimating with pomodoro (real time) rather than abstract points. We also explored the scenarios where the Pomodoro Technique should not be applied. If and how the technique can be applied in a distributed setting was also examined. Even though the Pomodoro technique is not used in a truly distributed manner in the case we studied (which is a major limitation of our study), we believe that the technique itself is a welcome addition to the agile development toolkit. We would hope that the learnings from this case study could be easily adapted into distributed settings.

Our study contributes to the body of knowledge of an under-developed theme within agile research: effective time management in fast-paced agile software development. The practical implication of our study is a better understanding of time management in agile software development, and several concrete suggestions of how to effectively apply the Pomodoro Technique and make the best out of it in an agile team working in a distributed setting. Our study is just the first step in the investigation of interesting phenomena arising from the application of the Pomodoro Technique in agile teams. Our report of the case is presented from the perspective of the agile team that used the Pomodoro Technique, and claimed it to be an effective approach. To further establish the effectiveness of this technique, more objective assessment is needed, and the viewpoints of all stakeholders related to the team need to be obtained. During this research, other issues emerged as candidates for future exploration, including how to encourage team members to take a break, how to use timers properly in an open office (visible vs. invisible timer, sound of ticking and ringing, and mechanical timer vs. digital timer vs. software tool), interesting secondary effects associated with breaks, and learning associated with using pomodoro for estimation.

Acknowledgements Special thanks go to Matteo Vaccari and his Sourcesense Milan Team who collaborated on our study and supported the production of this book chapter. Their experiences with the Pomodoro Technique were the inspiration of our study.

References

1. Ågerfalk, P. J., Fitzgerald, B., Holmstrom, H., Lings, B., Lundell, B., & Ó Conchúir, E. (2005). A framework for considering opportunities and threats in distributed software development. In *Proceedings of the international workshop on distributed software development (DiSD 2005)* (pp. 47–61), Paris, 29 August 2005. Vienna: Austrian Computer Society.
2. Agile Manifesto (2001). <http://www.agilemanifesto.org/>, last visit Nov. 2009.
3. Beck, K. (2000). *Extreme programming explained: Embrace change* (1st ed.). Upper Saddle River: Addison-Wesley.
4. Brooks, F. P. (1975). *The mythical man-month—Essays on software engineering*. Upper Saddle River: Addison-Wesley.
5. Cirillo, F. *The pomodoro technique* (XPLabs Technical Report version 1.3). English Version. <http://www.tecnicadelpomodoro.it>. Published 15 Jun 2007.
6. Cramton, C. (2001). The mutual knowledge problem and its consequences for dispersed collaboration. *Organization Science*, 12(3), 346–371.

7. Espinosa, J. A., Cummings, J. N., Wilson, J. M., & Pearce, B. M. (2003). Team boundary issues across multiple global firms. *Journal of Management Information Systems*, 19(4), 157–190.
8. Grinter, R. E., Herbsleb, J. D., & Perry, D. E. (1999). The geography of coordination: Dealing with distance in R&D work. In *Proc. int'l ACM SIGGROUP conf. supporting group work (GROUP '99)* (pp. 306–315). New York: ACM Press.
9. Gobbo, F., & Vaccari, M. (2008). The pomodoro technique for sustainable pace in extreme programming teams. In *Proceedings of XP2008*, Limerick, June 2008.
10. Hazzan, O., & Dubinsky, Y. (2007). The software engineering timeline: A time management perspective. In *Software-science, technology & engineering, 2007. SwSTE 2007. IEEE international conference on* (pp. 95–103). Herzlia, Israel.
11. Humphrey, W. S. (2000). *Introduction to the team software process. SEI series in software engineering*. Upper Saddle River: Addison-Wesley.
12. PMI (2004). *A guide to the project management body of knowledge (PMBOK)*.
13. Sarker, S., & Sahay, S. (2004). Implications of space and time for distributed work: An interpretive study of US-Norwegian systems development teams. *European Journal of Information Systems*, 13(1), 3–20.

Chapter 11

MBTA: Management By Timeshifting Around

Erran Carmel

Abstract How do good managers manage and coordinate? As technologies evolve the answer has also been evolving—from MBWA (Management By Wandering Around), to MBFA (Management By Flying Around), and now to MBTA (Management By Timeshifting Around). The purpose of this chapter is to surface and introduce this de-facto managerial approach.

11.1 Management by Wandering and Flying Around

How do good managers stay in touch? How do they coordinate? How do they motivate? As technologies evolve, the answers to these questions are also evolving. The purpose of this chapter is to surface a de-facto managerial approach.

One of the perennial management problems is that fallible managers become complacent and lose touch. In traditional hierarchical organizations, managers risk losing touch by relying on layers of intermediaries. In the age of information systems, managers may lose touch by relying entirely on computer generated data.

Hence, MBWA: Management By Wandering Around. MBWA is a management “method” relying on interpersonal (face to face) contact. The manager visits with the workers, often without notice. MBWA managers are widely thought to be more effective than managers who do not wander around.

MBWA can achieve a myriad of positive process goals for managers: keep abreast of operational progress and problems, actively listen to and engage with employees, learn soft information, get to know what employees are thinking and what they are up to. MBWA also lets employees feel that the manager cares and is available, and gives opportunities to managers to motivate employees by repeating key messages.

E. Carmel (✉)
American University, Washington, DC, USA
e-mail: carmel@american.edu

MBWA, also known as Management By Walking Around, was first publicized by HP's David Packard in the 1940s [1]. It was revived and became broadly recognized due to the highly influential book *In Search of Excellence* [2].

With the dawn of widespread distributed/virtual organizations and teams in the 1990s, MBWA was clearly impossible and had to be substituted with other management approaches. In 1999 I wrote about MBFA, Management by Flying Around [3]. I did not coin this term, though my book played an important role in publicizing it. I learned this new acronym from pioneering virtual managers at IBM and observed it in the behavior of other managers that I interviewed in the 1990s.

At that time, in the 1990s, we generally believed that the dominant work configuration was that of *clustered* locations in which co-located programmers work together in an office (e.g., the American programmers were in an office building in Reston and the Irish testers in an office building in Stillorgan, and the Swedish programmers in Kista, the Chinese programmers in a skyscraper in Pudong, and so on). Therefore managers were able to fly from location to location in order to meet with and work with their programmers.¹

One decade later there are now several million knowledge workers (not just programmers) involved in international distributed work groups. Technology has become much friendlier since the 1990s. However, travel is prohibitive for most of these teams and thus, MBFA is impossible.

11.2 Enter Timeshifting

Instead of MBFA, what one finds nearly everywhere today is **MBTA, Management By Timeshifting Around**. Managers stay in place, but timeshift to different locations by adjusting or scattering their work day. Timeshifting is defined as adjusting one's work hours to accommodate another's schedule.

Typical of MBTA is one company I studied in 2009. This American SoC (System On a Chip) company has R&D centers in France (GMT +1) and China (GMT +8). Yet the CTO was living in California (GMT -8) (See 11.1). The CTO, working out of his California home office, addressed the overlap time by breaking up his daily work routine into two work shifts. The first was in the morning to be available to the French engineers and the second shift was in the late afternoon to be available to Chinese engineers. He used the middle of the day to exercise, take care of errands, or catch a quick siesta. In addition to the timeshifting of the CTO, each of the two distant R&D sites relied on 3–5 people that do some timeshifting to communicate on a regular basis.

MBTA is not necessarily prescriptive, but rather descriptive: it is commonly practiced. Conceptualization of MBTA came about from more than a decade of

¹In a corollary I culled key management attributes for such managers [3]. I combined them into 5 special characteristics that the breed of global managers needs to have, represented by the acronym MERIT: Multi-culturalist, E-Facilitator, Recognition promoter, Internationalist, Traveler. The last two are closely linked to MBFA.

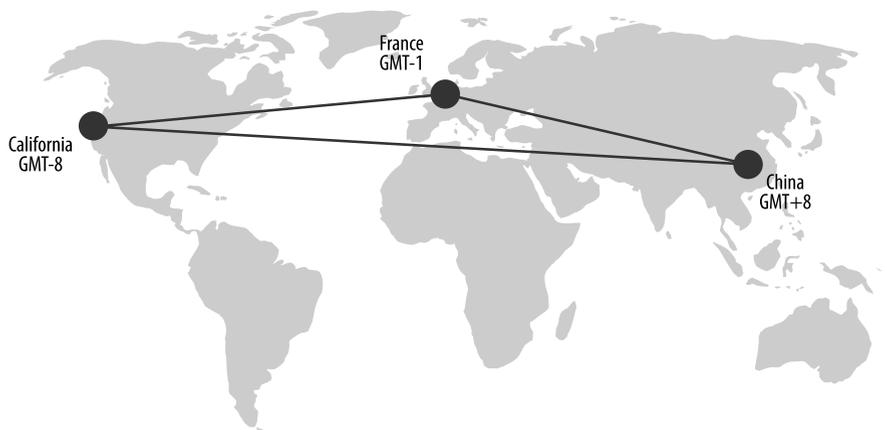


Fig. 11.1 Locations involved in timeshifting

research, which includes hundreds of formal and informal interviews with managers all over the world (this research is conducted, in part, with my colleague Alberto Espinosa). In these time zone-separated work groups we have found one management/coordination solution prevalent nearly everywhere: *timeshifting*. A careful reading of most distributed agile cases and best practices will uncover at least some timeshifting [cf. [4]].

MBTA managers can potentially achieve many of the advantages of MBWA and MBFA. They can have one-on-one intimate conversations via Skype or their mobile telephone. They can work “together” with others examining work products. They can develop key personal relationships, trust, common mental models and other important process attributes. Or, they can just share a joke.

The managers that conduct MBTA are often the *liaisons* [3]. A liaison is a person who bridges time zones and culture. The liaison is typically a mid-level manager, such as a project manager. He/she is the one who stays up late at night to make the critical telephone calls. He/she is the one who is able to speak across cultures and across languages. The liaison goes by varied labels. For example, at Microsoft, in distributed agile development, the liaison may be the team-room buddy [4].

Of course, there are other coordination tactics besides MBTA that are used in order to overcome time zones differences. One useful framework is to categorize these tactics into mechanistic, organic and implicit [5]. Mechanistic tactics are structured and routine (e.g., methodologies). Organic tactics are ad hoc and usually involve some conversation or interaction (thus, in time zone distributed teams, organic coordination using asynchronous email tends to be problematic because in complex tasks, messages frequently require clarification when the other party is sleeping). Implicit tactics have to do with various kinds of mental models that are shared across team members.

MBTA can be seen as encompassing elements of all three of the above coordination tactics. It is mechanistic if it is routine; it is organic in that MBTA is centered

around verbal conversations; MBTA is implicit in that the team members' mental models of each other are improved.

My data on MBTA are not specific to agile teams. However, given that many agile projects are distributed and substantially time-zone challenged, then timeshifting, or more specifically, MBTA, is unavoidable. For distributed agile development MBTA is useful and necessary: it leads to informal unstructured discussions. And, it facilitates the important standup meeting.

11.3 Conclusions

A key implication for the agile and distributed community is that the project stakeholders need to recognize that MBTA is necessary. They must do two somewhat contradictory things: they must demand that agile members timeshift and they must demand that all be sensitive to the personal hardship that this takes on their personal lives.

References

1. Hoover, J., & DiSilvestro, R. P. (2005). *The art of constructive confrontation: How to achieve more accountability with less conflict*. New York: Wiley.
2. Peters, T., & Waterman, R. (1982). *In search of excellence: Lessons from America's best run companies*. New York: Harper and Row.
3. Carmel, E. (1999). *Global software teams: Collaborating across borders and time zones*. Upper Saddle River: Prentice Hall.
4. Miller, A. (2009). Distributed agile development: Experiments at Microsoft. In *Proceedings of Agile 2009*. Also available as a Microsoft white paper: <http://www.pnpguidance.net/Post/DistributedAgileDevelopmentMicrosoftPatternsPractices.aspx>.
5. Espinosa, J. A., Slaughter, S. A., Kraut, R., & Herbsleb, J. (2007). Team knowledge and coordination in geographically distributed software development. *Journal of Management Information Systems*, 24(1), 135–169.

Chapter 12

The Dilemma of High Level Planning in Distributed Agile Software Projects: An Action Research Study in a Danish Bank

Per Svejvig and Ann-Dorte Fladkjær Nielsen

Abstract The chapter reports on an action research study with the aim to design a high level planning process in distributed and co-located software projects based on agile methods. The main contributions are the insight that high level planning process is highly integrated with other project disciplines and specific steps has to be taken to apply the process in distributed projects; and the action research approach is indeed suitable to software process improvements.

12.1 Introduction

For several years many organizations have taken advantage of doing distributed software development projects, involving cooperation and collaboration between teams located at different locations [1]. Reduced cost is a main driver for moving into these distributed projects, but another reason is higher flexibility of competences and resources [2] with talented knowledge workers around the world. To manage teams in different locations is a great challenge due to asynchrony of communication, reduced opportunities for rich interactions, difference in culture and work practices, and finally lack of trust [1, 3].

Agile methods were introduced in the late nineties in order to develop software quickly and efficiently [1]. They became popular because they treat requirements as emergent and volatile during the development process thereby acting to rapid

P. Svejvig (✉)
Aarhus School of Business, Aarhus University, Århus, Denmark
e-mail: psve@asb.dk

A.-D. Fladkjær Nielsen
Jyske Bank, Silkeborg, Denmark
e-mail: afn@jyskebank.dk

changes in organizations, something that traditional software methods have lacked. Agile methods are furthermore based on frequent face-to-face interactions in small teams producing small software products [4]. Applying agile methods in distributed software projects is a challenging cocktail, because the nature of agile projects with co-located team members is quite the opposite of that of distributed teams located at different locations [1]. As a consequence, organizations choosing this approach need to carefully consider how to implement an appropriate software development process. We consider agile methods as practices and tools for agility in teams while agile projects consists of teams of people, often co-located, applying these agile methods for their work.

Distributed as well as co-located software projects that apply agile methods, have the mantra “*responding to change over following a plan*” [4] which may serve as an excuse for developers to neglect the planning, which is one of the problems that Jyske Bank, a Danish mid-sized bank, is experiencing in their distributed and co-located agile software projects. Developers are claiming that high level planning is difficult in agile projects due to the following: (1) high level planning is conflicting with the agile manifesto [5], and (2) they “feel” that to prepare high level plans is like following the old waterfall approach, tagging them as old-fashioned.

The task of making a high level plan (HLP) ought to be done in every project. It is a discipline which many plan driven projects are good at. However in agile projects which furthermore are manned distributed this becomes much more difficult, because the need for high flexibility regarding scope in the agile process makes it hard to plan on paper. Furthermore when dealing with distributed teams doing preparation and follow up on the HLP, it becomes quite difficult due to the physical separation.

The consequence is that steering committees, external parties and other stakeholders cannot get appropriate information about “what can be delivered at what times”, and the business, i.e. customers of the software projects, loses confidence in the IT organization. This is the dilemma presented in this chapter, which leads to our research question: *How to design a high level planning process in distributed and co-located software projects based on agile methods?*

To address this challenge we set out to undertake an action research study in several cycles, where this paper reports the findings from the first cycle. The main contributions are (1) the insight that high level planning process is highly integrated with other project disciplines and specific steps has to be taken to apply the process in distributed projects; and (2) the action research approach is indeed suitable to software process improvements.

The chapter is organized as follows. First we briefly introduce the action research methodology followed by the research setting describing the Jyske Bank case. Then we explain the four selected steps in the action research cycle as diagnosing, action planning, action taking and evaluation & learning. Finally, we conclude with practical advices based on the completed action research cycle.

12.2 Research Methodology

12.2.1 Action Research

To answer the research question we have undertaken an action research study in Jyske Bank from August to November 2009. Action research (AR) involves close cooperation between practitioners and researchers to bring about change [6]. The action research process can be defined as a number of learning cycles consisting of predefined stages, as presented in Fig. 12.1 below [adapted from [7, 8]]:

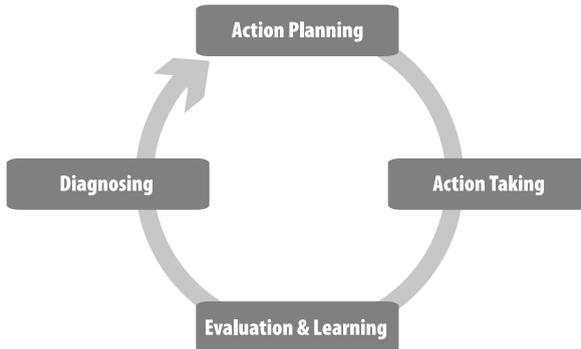


Fig. 12.1 Action research process

The AR cycle starts with *diagnosing*, which refers to the joint (practitioner and researcher) identification of problems and their possible underlying causes. *Action planning* specifies the anticipated actions that can improve or solve the problems. *Action taking* refers to the implementation of the specified actions. *Evaluating* is the assessment of the intervention, and finally *learning* is reflection on activities and outcome [ibid.]. The chapter describes results from the first AR cycle while two cycles are needed to design and implement the final high level planning process.

Empirical data was gathered through joint workshops (practitioners and researcher), interviews, participant observations, informal meetings, informal communications (e-mails) and documents from the organization and specific projects.

12.2.2 Research Settings

The Jyske Bank group is a financial institution that provides all types of financial services such as banking and financial deals primarily in Denmark. Jyske Bank employs 3,800 employees and has more than 500,000 private and business customers in Denmark. As part of the Jyske Bank Group there is an IT division with 400 employees located at the headquarters in Denmark. IT projects at Jyske Bank vary widely in size; most are small and short term, but there are also large projects that have strategic implications for the entire Jyske Bank group. Typically project teams of three to twenty people handle the small projects, which usually take from six to twelve months. These projects are the main focus of this AR study.

Table 12.1 Company overview

Jyske Bank	
Number of developers	400
When was agile introduced	2006
Domain	Financial institution with Banking

The IT division is headed by a Senior Vice President and is organized into departments with typically 20 to 40 people, working on two or more projects. Project Managers oversee regular projects, while Senior Project Managers have the responsibility for high-profile projects.

Jyske Bank started in 2006 to use agile methods based on Scrum [9] which has however been adjusted to the Jyske Bank organization. The most essential changes are as follow.

Table 12.2 Jyske Bank changes to Scrum

SCRUM	Jyske Bank Changes
The customer determines the overall goals and constrains for projects	The steering committee determines the overall goals and constrains for projects because there are only internal customers
The iterative development starts day one	The iterative development normally starts after high level design with a duration of two to six weeks
Assign Scrum master	Project manager fulfills the role as Scrum master
Assign product owner	The business responsible, architecture responsible and project manager are jointly fulfilling the role as product owner

First Jyske Bank did only work with agile methods in relation to co-located projects, but in 2008 they started to work with distributed projects as well (Denmark and India). We understand co-located projects as one or more teams located at the same physical work place while distributed projects implies that one or more teams are spread across two or more separate physical work places.

12.3 The Action Research Cycle

12.3.1 Diagnosing the Problem and the Underlying Causes

There have been several challenges associated with the introduction of the agile methods including Scrum: (1) to play the roles in the project; (2) to focus on producing smaller products, and (3) finally to make the interaction between iteration planning and high level planning work. The first two points deal with change management in IT projects, and here the organization has got very far. The last point

has so far been a “trial and error” process within the projects and with great variance across the projects. The projects have attempted to make plans without using templates. They have only been guided by the stakeholder requests regarding timing overview of the project. This is certainly not good enough so the IT management decided that a concept should be developed to support the projects, and enable a standardized approach to high level planning.

An AR task force was established with the purpose “to enable that IT projects create an overview of the entire project assignment in the form of a high level plan to be used as a means of communication towards the steering committee, external partners and other stakeholders”. The problem was recognized as lack of high level planning and the underlying causes were understood to be resistance against high level planning in agile projects according to IT management.

The scope for the AR task force is at the project level, as presented in Fig. 12.2 below.



Fig. 12.2 Scope for the action research process

12.3.2 Action Planning

The AR task force decided to use *semi-agile methods* for developing and implementing the HLP concept for instance with a high degree of “customer collaboration” [5], which were combined with two action research cycles according to the plan in Fig. 12.3.

Figure 12.3 shows the main activities in the two AR cycles. The first AR cycle is completed and reported in this book chapter while the next AR cycle is planned for spring 2010.

12.3.3 Action Taking

The process was started by a two-day joint workshop in August 2009 involving four projects (two distributed and two co-located). The first day was allocated to dis-

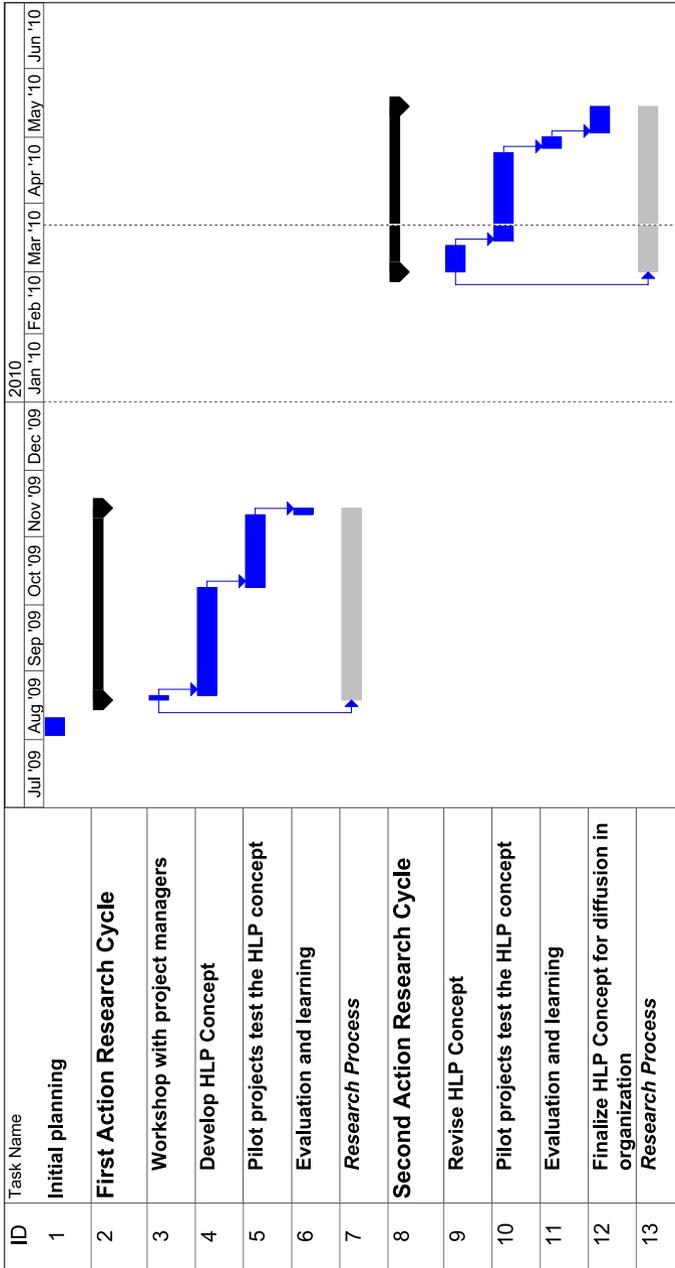


Fig. 12.3 Action research plan

cuss how the four projects deal with high level and iteration planning today (“AS IS”), and the second day was devoted to discussing the future high level planning process (“TO BE”). Seven persons participated in the workshop: four project managers representing four projects, two facilitators (the leader and assistant from the software engineering process group) and a researcher (intervention and inspiration from theory/practice). The workshop was intense and highly participatory—below is an annotated photo from one of the project presentations:

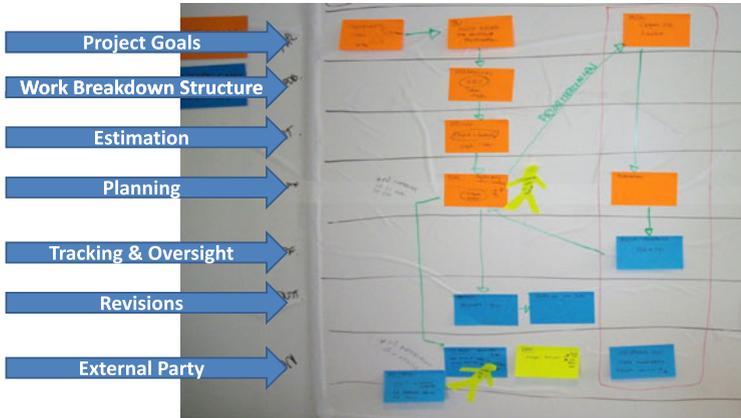


Fig. 12.4 Example of “AS IS” process from a project

Important results from the two-day workshop are: (1) a HLP process should be applicable to both agile and plan-driven projects; (2) most projects have both agile and plan-driven elements; (3) HLP cannot be isolated from estimation, risk analysis, stakeholder analysis etc.; and finally (4) different stakeholders have different needs related to the HLP process. The second day a “high level plan prototype” was developed to be used in the further process.

There were two surprising outcomes from the workshop. First, the IT project managers were highly motivated for the HLP process and not resistant against the HLP process although they expressed the need for a solid and standardized procedure for managing the HLP process, because they find the HLP process difficult. Second, it is neither possible nor desirable to isolate the HLP process as a “stand-alone process”, but instead to design a more holistic and coherent process including integration with high level estimation process, format for communication to the steering committee, and other related topics.

The prototype was revised by the AR task force and discussed with key stakeholders in three interviews. A line manager states that the current high level planning process is unstructured and not standardized and argues that estimation expertise is an important and inevitable ingredient of the HLP process. The line manager considers the HLP plan both as “a written plan and a verbal presentation of the plan”. The verbal part is the communication going on at steering committee meetings and other meetings mediated by presentations, plans, documents etc. All formal and informal

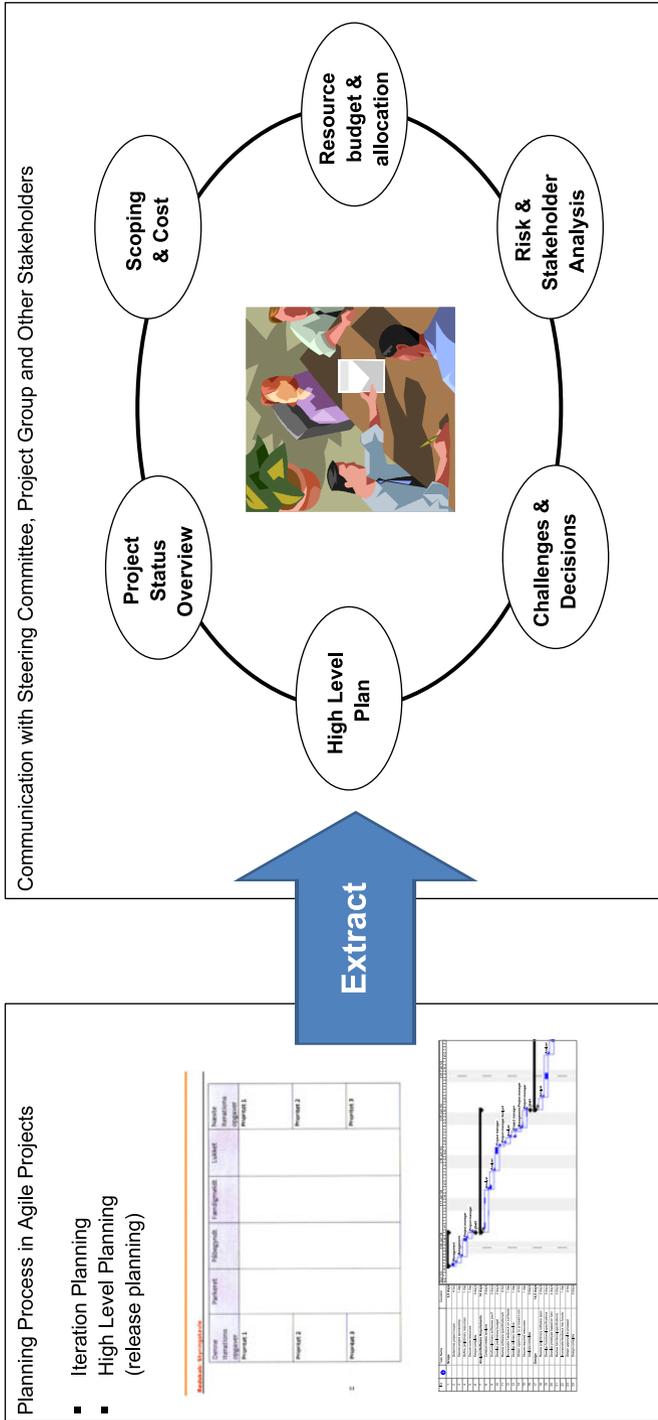


Fig. 12.5 The high level planning concept model

comments were considered and the prototype was revised accordingly. A simplified representation of the prototype is shown in Fig. 12.5.

Two projects applying agile methods were selected as pilot projects for the first AR cycle. The first one is a co-located project about calculation of customer ratings and the other a distributed project for data reorganization related to the data warehouse. Both projects have used the new HLP concept at steering committee level and at project group level.

Table 12.3 Calculation of customer ratings (Project 1)

Duration	18 months
Status	Ongoing
Agile practices	Jyske Banks methods based on Scrum
Involved locations	DK (co-located)
Numbers of participants	8
Roles	Project manager, Business responsible person, Architecture responsible person, IT developers, Business consultants

Table 12.4 Data reorganization related to data warehouse (Project 2)

Duration	20 months
Status	Ongoing
Agile practices	Jyske Banks methods based on Scrum
Involved locations	DK, India (distributed)
Numbers of participants	11 in DK and 3 in India
Roles	Project manager, Business responsible person, Architecture responsible person, IT-developers, Business consultants in DK and 3 IT developers in India

In the two pilot projects the project manager has taken the initiative and been responsible for the HLP. The project manager has involved the business- and architecture responsible persons in the work, and introduced a draft which the rest of the project group has enriched further with their knowledge.

In the distributed project the developers from India was included in a video session in which the rest of the project group also were present in Denmark.

We have been actively involved in the two projects by supporting preparation of steering committee presentations, participation in project meetings and steering committee meetings, and interviews with the project managers. We have got feedback from the project managers and given them feedback as well.

12.3.4 Evaluating and Learning

The project managers from the two pilot projects were positive towards the HLP concept. Feedback from the project managers and steering committees about using the HLP concept are shown below:

Table 12.5 Feedback on high level planning concept

Project 1—Co-Located	Project 2—Distributed
Use of traffic light to indicate the status of project and sub-projects is a good way to communicate the perception of the project status	The template is a sound way to streamline project presentations at steering committee meetings
Resource budget & allocation overview have to be at an appropriate level of detail	Resource budget & allocation overview is very useful, but different steering committee members request different levels of details
The template for the steering committee meeting serves as a good check list where project managers can select the elements, which are relevant for the specific project	Re-estimation and re-planning (revised HLP) should be done after each iteration
Focus on cost management is relevant	The HLP concept is useful to steering committee meetings, but more problematic to use in the virtual interaction with the team in India (e.g. presentation of a four page MS-Project plan is easier on a notice board than via a web cam with medium resolution)
Cost management and risk analysis should be mandatory	

The comments in the table above reflect the emergent understanding in the AR task force about the HLP process as part of a more holistic and coherent overall process including communication with steering committees and other stakeholders.

The distributed challenges associated with the HLP process and communication more broadly deserve more detailed evaluation, and this has been discussed thoroughly with the project manager of the distributed project. The Indian team was not involved in the initial HLP process, because they had limited domain knowledge and was therefore not able to contribute. However the involvement of the Indian team on a more daily basis is necessary, but also problematic—the project manager says,

The communication of the plan works much better if we can physically place it on the wall and examine it. The times we have tried to review the plan electronically, it is my impression that the result has been limited. That applies both in Denmark and India. Since there are limits to how often we visit the Indians or they visit us, it is limited, how much they actually get out of the produced plans.

So it is an unresolved challenge from this first AR cycle how to involve distributed teams in producing, reviewing and discussing high level plans. The project manager

points at better technology as one step to overcome the problem (i.e. high resolution video, telepresence equipment etc.), but it might also be related to work practices.

The overall learning from the first AR cycle was the unanticipated turning from strict focus on high level planning to steering committee communication including high level planning. High level planning is still a key issue for the projects, but as an integral part of other project disciplines like communication, estimation, staffing, risk analysis etc. and it should not be separated from these disciplines.

The learning from the first cycle will be used to revise the HLP concept for the next AR cycle. We have decided to follow the two pilot projects above in the next AR cycle, but we will also add more projects to broaden the concept and bring more practical experience into the process. We will furthermore focus on good examples of plans and presentations for both co-located and distributed projects prepared by the project managers of the pilot projects.

12.4 Conclusions

The action research study described in this chapter was initiated by Jyske Bank who experienced problems with high level planning in distributed and co-located software projects applying agile methods. The first AR cycle is completed and implied a broader scope of the high level planning process than originally anticipated. We have identified some practical advices from this first AR cycle, which are described in the following.

12.4.1 *Applying a Holistic Approach to High Level Planning*

First, diagnose the problem carefully before prescribing the medicine. The discourse of the HLP process in Jyske Bank (mainly driven by IT management) was that “project managers are resistant against high level planning in distributed and co-located projects applying agile methods”. But the AR task force has never met this resistance in workshops, interviews etc., so the discourse did not reflect the situation, and we had to prescribe another medicine. However the project manager expressed concerns about the complexity in the HLP process, but this is not resistance. This advice is applicable to other software process improvement areas and requires that practitioners (and researchers) better understand the problem before aiming at to solve the problem.

Second, consider the high level planning processes as broader than strict planning. HLP is an integrated part of the project management process and belonging disciplines and involves especially communication of high level plans to stakeholders such as steering committee and external parties.

Finally, design the distributed approach into the process. Software engineering groups designing and implementing project templates and processes have to consider both co-located and distributed projects. There is a tendency to understand the

project processes locally and to design templates and processes accordingly. The challenge with distributed projects has been toned down at Jyske Bank, but the distributed aspect has to be an essential ingredient in designing project templates and processes targeting agile methods and project management in general.

12.4.2 Using Action Research to Software Process Improvement

The action research approach with semi agile methods is a promising way to design project processes and templates. This includes the high degree of customer collaboration, which has been important and necessary in order to design workable processes and templates. This is also a nice way to break down barriers between the people executing projects (project managers, project participants, steering committee members etc.) and the software engineering process group designing the project processes and templates.

12.4.3 Summary

The chapter reports on an action research study with the aim to design a high level planning process in distributed and co-located software projects based on agile methods. The chapter reports from the first action research cycle and the main contributions are the insight that high level planning process is highly integrated with other project disciplines and specific steps has to be taken to apply the process in distributed projects; and the action research approach is indeed suitable to software process improvements.

References

1. Bose, I. (2008). Lessons learned from distributed agile software projects: A case-based analysis. *Communications of the Association for Information Systems*, 23, 619–632.
2. Dibbern, J. et al. (2004). Information systems outsourcing: A survey and analysis of the literature. *Database for Advances in Information Systems*, 35(4), 6–102.
3. Staples, D. S., & Webster, J. (2008). Exploring the effects of trust, task interdependence and virtualness on knowledge sharing in teams. *Information Systems Journal*, 18(6), 617–640.
4. Boehm, B. (2002). Get ready for agile methods, with care. *Computer*, 35(1), 64–69.
5. Beck, K. et al. (2001). Agile software development manifesto. Available via: <http://agilemanifesto.org/>. Accessed 1st July 2009.
6. Gray, D. E. (2004). *Doing research in the real world*. London: SAGE Publications.
7. Grant, D., & Ngwenyama, O. (2003). A report on the use of action research to evaluate a manufacturing information systems development methodology in a company. *Information Systems Journal*, 13(1), 21–35.
8. Davison, R. M., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, 14(1), 65–86.
9. Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). Using scrum in a globally distributed project: A case study. *Software Process Improvement and Practice*, 13(6), 527–544.

Chapter 13

Tools for Supporting Distributed Agile Project Planning

Xin Wang, Frank Maurer, Robert Morgan,
and Josyleuda Oliveira

Abstract Agile project planning plays an important part in agile software development. In distributed settings, project planning is severely impacted by the lack of face-to-face communication and the inability to share paper index cards amongst all meeting participants. To address these issues, several distributed agile planning tools were developed. The tools vary in features, functions and running platforms. In this chapter, we first summarize the requirements for distributed agile planning. Then we give an overview on existing agile planning tools. We also evaluate existing tools based on tool requirements. Finally, we present some practical advices for both designers and users of distributed agile planning tools.

13.1 Introduction

Agile project planning is an important activity for agile teams. It allows a team to start focusing on the next development iteration and drives the evolution of software products. The goals of agile project planning include:

X. Wang (✉)
Ivrnet Inc., Calgary, Alberta, Canada
e-mail: x.wang@ivrnet.com

F. Maurer · J. Oliveira
Department of Computer Science, University of Calgary, Calgary, Alberta, Canada

F. Maurer
e-mail: maurer@cpsc.ucalgary.ca

J. Oliveira
e-mail: oliveirj@cpsc.ucalgary.ca

R. Morgan
Red Duck Solutions, Calgary, Alberta, Canada
e-mail: robert.morgan@redducksolutions.com

- Controlling the software development progress.
- Kicking off a new development iteration and planning tasks for it.
- Providing a focal point for the communications between developers and customers.
- Enhancing the collaborations within software development teams.

Project planning benefits agile development in both management and communication aspects: it reviews project progress, detects development bottlenecks and generates sound plans to decide on the use of team resources. It connects developers with customers and reduces the misunderstandings between each other.

Traditionally, agile planning is conducted in a co-located environment. Participants are situated at the same site, using face-to-face communication to plan future tasks. Index cards are the major artefact that supports the project planning process. New tasks are created by writing new cards. Tasks are prioritized by sorting the cards.

When we observed co-located agile project planning meetings, we found three primary factors that positively affect their quality:

- *shared access index cards describing tasks,*
- *flexible use of index cards and*
- *easy interactions among meeting participants.*

The shared access to index cards enables participants to understand the current state of the planning process. The flexibility of using index cards helps developers plan the project in a convenient manner. Easy interactions among participants improve collaboration among all stakeholders. The collaborative environment improves the effectiveness of project planning and help shape the group into a unified and well-communicating team.

In a co-located environment, all three primary factors are easily provided. Paper index cards on a table provide an intuitive and shared access to project tasks. Physical cards are easily edited or ranked. Natural interactions are the result of verbal communications and body gestures.

However, when agile teams are distributed it is difficult to conduct traditional agile project planning meetings. Sharing planning artefacts among spatially-separated environments becomes challenging, and interaction among planning participants are more difficult. When we conducted interviews with distributed agile teams from Brazil and Canada, several issues were pointed out. These include:

- Making decisions becomes much harder than that in co-located project planning meetings.
- Sites were not talking like a unified team.
- There is less communications within a distributed team than a co-located one. Respectively, problems are not reported until they are bigger. Then, they require more time and money to solve them.
- Misunderstandings are raised and the chance of rework is increased significantly.

A first commonly used approach for distributed project planning is to utilize audio and/or video conferences with paper index cards at each site. Telephones and cameras are employed to set up synchronous verbal and visual communication

among different sites. Although such an approach establishes remote interactions, index cards are not shared in this context they either reside at one site only or are replicated manually at the other sites. Story cards from one site are not directly shown to the distributed teams, and key behaviours, such as modifying index cards are difficult to share with remote colleagues. When distributed teams replicate paper story cards to each site, the duplication increases the risk of misunderstandings. To understand the impact of distributed teams, we conducted a small scale experiment. We observed a meeting between teams in Canada and United Kingdom that used conference calls and replicated paper index cards for the meeting. The discussions were often interrupted by both teams realizing they were not talking about the same card. The meeting also ended with both sites generating a different number of index cards for the same topic, which represents a severe misunderstanding between both sites. Anecdotal evidence from practitioners confirm these findings.

As a result of these experiences, several attempts were made at using computers and the Internet to set up a card-centered project planning environment. A large number of tools are now available to support distributed agile planning. In this chapter, we start by discussing high-level requirements for distributed agile planning tools. We then review existing tools based on these requirements. Practical advice will be provided for users and designers of distributed planning tools to assist with selecting or designing an appropriate tool to support distributed agile planning.

13.2 Distributed Planning Tool Requirements

Our study started by setting up a series of requirements for distributed agile planning tools. The requirements are used as criteria to evaluate and compare existing tools. From these requirements, users can evaluate the benefits and limitations of tool usage within their teams. Designers will also understand from which aspects a distributed agile planning tool can be improved. In this section, we break down the tool requirements to those specific to agile planning and those specific to collaborative interactions:

- *Agile planning requirements:* Distributed agile planning tools are specifically designed to support agile developments. Therefore, the primary requirements are to cover the major functions prescribed by the agile planning processes such as creating and editing index cards. The agile planning requirements concentrated on the tools functional capabilities and determine whether and how much agile planning is supported. They are proposed from reviewing literature from previous research [1, 8] and our practical experiences on developing agile planning tools [2, 9, 15].
- *Requirements for collaborative interactions:* Agile project planning is essentially a group-based collaborative activity. It can be improved by providing easy interactions for team members. Unhindered interactions are also the main approach that makes distributed collaborations more effective. In order to improve the interactions among distributed groups, a substantial amount of research was conducted on computer supported collaborative work (CSCW) processes and groupware.

We believe the general studies on CSCW and groupware largely benefit the development of distributed agile planning tools. In this section, we referred to the literature on groupware research and proposed a series of requirements for collaborative interactions. The requirements show a higher level goal than just being able to do agile project planning. These requirements concentrate on tool usability and highlight the importance of supporting interpersonal interactions by enabling intuitive human-computer interaction for distributed agile planning. They also indicate how a distributed agile planning tool can be effective and convenient to use.

13.2.1 Agile Planning Requirements

The agile planning requirements stem from the need to create, organize and share planning information. Having observed distributed agile planning as well as analysing the related literature, we found the following requirements are critical to support distributed agile planning. The first three requirements are derived from previous studies made by Abrahamsson et al. [1] and Larman [8]. The remaining requirements are a result of our observations of planning meetings and experiences in developing distributed agile planning tools.

1. **Creating, editing and deleting planning objects.** The fundamental requirement of any agile planning tool is its ability to support the creation, modification, and deletion of planning artefacts. In a co-located scenario, a developer grabs an empty card and edits it to define a new task. He/she also can remove obsolete cards from the planning table. Remote agile teams still follow a card-centred planning process: given only an audio link, they create and manipulate index cards on their own site. Thus, operations for creating/editing/deleting cards are needed in any tool supporting distributed planning.
2. **Handle effort estimates.** Experience working on a project can be an important piece of information when trying to plan for the future. Keeping track of knowledge from previous iterations and story cards, such as estimates, priority, and actual effort, can be useful when trying to estimate new tasks. Managing this information can also be of use when determining the scope of current iterations (yesterday's weather).
3. **Planning multiple iterations.** Supporting multiple iterations when planning allows teams not only to plan at the iteration level but also to conduct long term release planning.
4. **Moving stories from one iteration to another.** Observing real-world agile teams has shown us practical cases where a story card is transferred to next iteration or moved into/out of the backlog. Respectively, distributed planning tools must provide a feature to support this behaviour.
5. **Authentication.** Security is important to prevent unauthorized access and modification to the information contained in the project plan.

6. **Real-time updates of the plan.** Remote access to the project artefacts is required such that, as changes occur, updated information is available on each participating site instantaneously.
7. **Visual characteristics for different types of stories.** Stories can often be broken down into distinguishable types like bug fixes, new features, changes to existing functionality or enhancements to name a few. Supporting a visual distinction between these different types of story cards is often used by teams.
8. **Integration with the development environment.** Planning tools are used by both the business side and the technical development side of the team. Supporting integration with the development environment increases access to the planning information for developers and makes it easier to keep the plan up to date for progress tracking.

13.2.2 Requirements for Collaborative Interactions

The following requirements outline considerations for how a distributed agile planning tool can support interpersonal interactions and enhance collaboration within distributed team members. We determine requirements based on contributions by groupware researchers [5, 6] and combine them with our observations of distributed agile project planning meetings.

1. **Fluid transition between individual and collaborative work.** Systems need to support distinguishing private data from public data.
2. **Telepointers for pointing and gesturing.** Telepointers are a groupware technology that uses a remote mouse pointer to represent mouse movement happening on other computers. Telepointers allow remote team members to point to specific index cards, thereby increasing the shared understanding of the current discussion. They also allow teams to follow interactions happening on remote displays.
3. **Real-time information sharing.** Sharing information requires that changes to one workspace be updated in other workspaces instantaneously.
4. **Change notification.** When changes to the workspace are made those changes need to be shared with the other team members regardless if they are connected to the plan or not.
5. **Joining and leaving meetings.** Team members should be able to connect and disconnect with ease and not affect others connected to the system.
6. **Fluid subgroup formation and dissolution.** For large scale projects consisting of teams of teams, subgroups need to be represented and supported by the planning tool. When supporting subgroup creation, the potential for isolation needs to be minimized in order for the subgroup to be informed of the other participants.
7. **Simultaneous interaction.** Supporting team members interacting simultaneously is important as it better simulates how teams interact in a co-located environment. Forcing teams to take turns would result in more overhead for the team during the planning meeting.

Table 13.1 Categories and sample tools

Category	Sample Tools
Wiki	MASE, PMWiki, JSPWiki, MediaWiki
Web-form based application	Rally, VersionOne, ScrumWorks, XPPlanner, Mingle
Board-based application	CardMeeting, Gluewiki, AgilePlanner, MASE, Mingle, Danube
Plugins for IDE	IBM Jazz, Jira+GreenHopper, ProjectCards
Synchronous agile planning tool	DAP, CardMeeting
Tabletop-based agile planning tool	APDT

13.3 Tool Review

Software systems to support agile project planning in distributed environments have been available for some time. Some tools focus on documenting the outcomes of a planning meeting for progress tracking during the iteration. Others target to support the actual planning meeting. Unfortunately, this difference is often not highlighted in the relevant literature and marketing material.

To review existing agile planning tools, we first collected candidate tools that are published online, mentioned by our interview participants (industrial agile developers), introduced by our partner companies or described in the literature. We reviewed the tools and found that although existing tools showed some diversity, they could still be categorized by design goals, functionalities and supported platforms. The categorizations help to reveal common features, advantages and limitations of existing agile planning tools. Table 13.1 lists the basic categories and sample tools in our study. Some tools are found sharing feature of more than one category.

13.3.1 Wikis

Wiki-based agile planning tools utilize Web technologies to publish, manage, integrate and distribute agile planning information. The advantage of using Wiki-based systems is that they provide a plain environment, making it easy to check project status, update task lists and view the team members' work progress. Wikis are an *asynchronous* platform for agile developers' communication and, thus, mostly helpful for progress tracking. The following scenarios show how an agile development team can use them:

- Publishing story cards [3]: After a project planning meeting, new wiki pages will be created to publish all the card information. Software developers and project managers will be able to access the wiki pages and check their tasks.
- Story card management: software developers are responsible for accessing the wiki pages and updating their cards every day. Updating the card status facilitates managing the development progress.

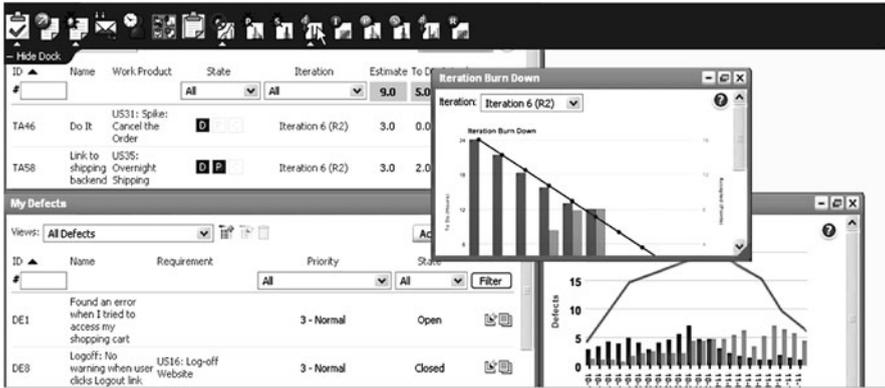


Fig. 13.1 Web form-based project planning tool [10]

- **Sharing knowledge:** a software developer can post his/her questions to a wiki, and his/her colleagues can view the questions to provide assistances. Meanwhile, one developer's experience of solving some critical problems can also be posted on the wiki to provide help to his/her teammates.

Wikis support asynchronous interactions for distributed teams. Using wiki pages does not rely on any specific software on the client side (any web browser will do). However, wikis only fulfill the minimum requirement of agile planning (creating/editing/deleting planning artifacts). Specific information of a user story, such as estimated hours, are mingled with plain text describing the story.

13.3.2 Web Form-Based Applications

Designers of distributed agile planning tools realized the advantages (easy access) and limitations (loosely organized planning data) of wiki pages and started to use advanced Web technologies to create a series of Web form-based applications. Compared with plain text wikis, the structured data stored by such tools supports more sophisticated functions and more flexible operations to manipulate agile planning information.

Web form-based applications are often used for publishing and managing agile planning data. Such tools include commercial products like Rally, VersionOne, ScrumWorks, as well as open source products like XPPlanner. These applications use Web forms to create and manipulate planning data. They also set up basic workflows for sharing data amongst distributed agile developers. Figure 13.1 shows a screenshot of the Rally tool.

Figure 13.1 shows that agile planning data is more structured by Rally than in wiki. Using the tool, users can change the status of a story card by clicking the status button. They can also update the estimated work hours or descriptions of a

task. Amongst other features, the Rally tool generates a burn down chart to help project managers and team developers understand progress of their projects.

The Rally tool shows some common features of Web form-based agile planning tools. Creating, editing and deleting story cards is supported. Charts are widely used to visualize the project progress. Agile planning data is well organized in projects, iterations, the backlog and story cards. Moreover, Web form-based applications can distinguish the roles of the users (such as developers or managers) and generate appropriate views for different user groups. The structured data managed in such tools provide semantically richer views on the agile process than text stored in wikis.

As Web technology is mature and the Web access is easily accepted by users, Web form-based applications dominate existing agile planning tools. However, most Web form-based tools are only for asynchronous usage (asynchronous data sharing, reporting, decision making, and daily card management), the synchronous agile planning, particularly the project planning meetings are not supported.

13.3.3 Card-Based Planning Systems

Card-based planning systems are systems that use visual representations that resemble index cards for representing tasks. These types of systems try to mimic physical card based planning. Glue Wiki, CardMeeting and AgilePlanner, amongst others, fall into this category. Several commercial agile planning tools, such as Thoughtworks Mingle, integrate a card-based planning with form-based tools. The benefit that card-based systems bring to planning is that they allow teams to interact with the cards as they are used from co-located settings.

Mingle (Fig. 13.2) uses two-dimensional representations of index cards that teams can edit and organize like paper index cards. It uses a browser to allow team members from any location to interact with the cards. Individuals are able to create cards and organize them spatially.

Card-based planning systems explore the collaborative interactions in distributed agile planning. The designs expect that showing the visual effects of “paper index card” might help agile teams adopting the tools. Card-based planning systems rely on spatial layout to make the current plan easier to understand. However, existing card-based planning tools (CardMeeting, Danube) do not show who is participating in the planning meeting. They specifically do not show if and who is currently interacting with planning artifacts. Knowing who is interacting with a planning artifact is important as it encourages communication and collaboration.

13.3.4 Plugin for Integrated Development Environment

Integrating project planning tools with Integrated Development Environment (IDE) will provide software developers a convenient environment for managing both codes



Fig. 13.2 Card-based system in Thoughtworks Mingle [13]

and planning data, such as story cards. At present, we observed two types of plugins. The majorities are repeating the major functions of native/Web-based project planning tools, which allow for browsing and editing project planning data. While another type of plugin, besides showing and managing planning data, have tried to find and utilize the relation between user stories and the practical working codes. It enables users to connect high level story cards with low level test cases and codes. It bridges the logical gaps between the user requirements with developers' implementation. IBM Jazz [7] explores integrating project planning tools with software developing platform. It provides an Eclipse-based client to enable software developers mapping their story cards with specific source codes. Navigation between the codes and cards are also provided. Besides IBM Jazz, Microsoft Visual Studio Team System (VSTS) is also interested in introducing the project planning plugins to Visual Studio platforms. Other related project planning tools includes "Scurm for Team System", "Jira + GreenHopper", and "ProjectCards".

13.3.5 Synchronous Project Planning Tool

Although Web technologies allow sharing of agile planning data, the features of agile planning tools are still limited by the capabilities of Web browsers. Several requirements of collaborative interactions, such as synchronous notifications, and using telepointers for pointing and gestures, are not fulfilled. Moreover, by reviewing the practical needs of industrial agile developers, we found that synchronous agile planning meetings are weakly supported by such tools.

Distributed Agile Planner (DAP) [9] is a standalone application for synchronous, card-based agile planning meetings. DAP mimics paper index cards. It simulates a whiteboard in a meeting room and utilizes electronic index cards to simulate paper

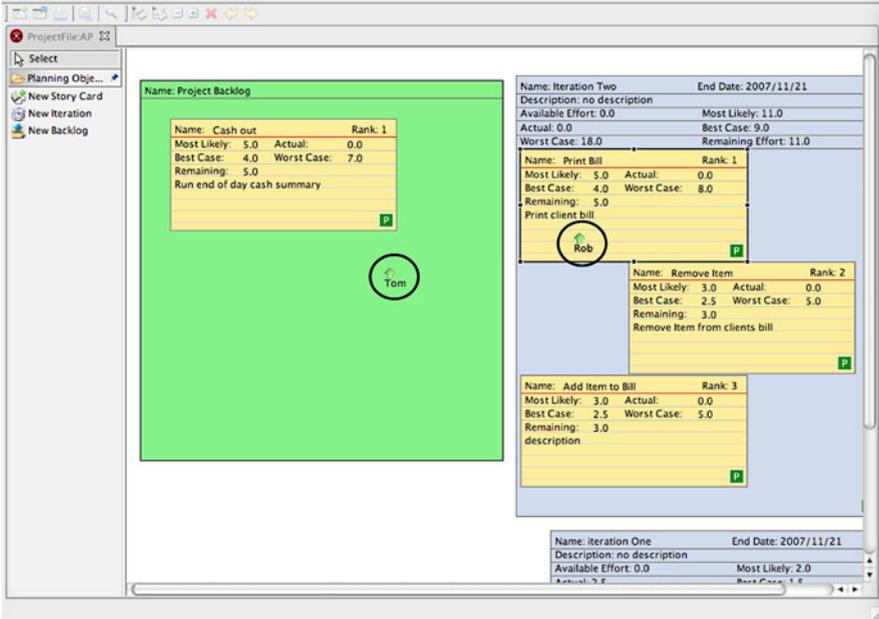


Fig. 13.3 Distributed agile planner with telepointers and digital cards

index cards. The user interactions of DAP include creating, moving and deleting cards. To support distributed collaboration, DAP provides telepointers [4] to represent mouse pointers of remote clients. The position of a tele-pointer is updated in real-time. Thus, a collaborator can understand his/her remote partner's mouse movement just like looking at hand movements in a traditional co-located meeting. Figure 13.3 shows the interface of DAP, on which a set of story cards and iterations are displayed. The green arrow is the "telepointer" which acts as a remote mouse pointer to indicate the focus of remote collaborations. DAP concentrates on the interactive collaboration but has only limited capabilities for progress tracking during the iteration. DAP is more primarily used for conducting the real-time planning meetings during which interactive collaborations are intensively observed. When we evaluated DAP during distributed planning meetings, we noticed a substantial change in interactions between participants at each site: most of the time, they looked at the shared screen instead of looking at each other. We believe that this is the result of putting a PC projector into the room and not a result of DAP. I.e. all other tools will suffer the same effect. Having a screen at the front of the room and assigning a single person in that room to control the mouse and keyboard, changes the social interactions between team members. When we observed DAP-based planning meetings, participants seemed to be less engaged in the planning process than in traditional agile planning meetings.



Fig. 13.4 Writing a digital card and the scenario of distributed teams sharing an APDT interface for their agile planning

13.3.6 Digital Tabletop-Based Agile Planning Tool

Digital tabletops [12] are novel user interaction devices. It has a horizontal display (a table that IS a computer display) and a multi-touch enabled surface to support concurrent touch-based interactions with that display. Agile Planner for Digital Tabletop (APDT) explores using digital tabletops for supporting distributed agile planning. It employs the interactive features of tabletops to enhance the user experience during distributed agile planning meetings. Using touch-sensitive interfaces and a handwriting recognition engine, APDT implements handwriting functions to simulate writing on a paper-based story card. As the tabletop has a horizontal and tangible screen, participants can sit or stand around table, using stylus or fingers to touch the virtual cards on the table surface (see Fig. 13.4). The multi-touch capability enables APDT users to concurrently interact with story cards without being hindered by each other. Telepointers are used to display touch interactions from remote sites.

An advantage of APDT over project planning tools using vertical displays/PC-projectors is that it simulates the co-located project planning and supports several user behaviors that were lost by using the traditional PC-based tools. APDT allows for using pens to write story cards, passing cards on the table surface, using a finger for dragging a card to a participant's territory and reorienting cards. The limitation of using APDT is that present tabletops are not widely available in industry. Only a small number of tabletops are commercially available (e.g. Microsoft Surface and SMART Table) and the purchase price is substantially higher than a PC projector. However, we believe that tabletops will become available in many industrial settings in the future.

13.4 Tool Evaluation

We now will evaluate agile planning tools based on the requirements for distributed agile planning identified above. The results are shown in Fig. 13.5. In this figure, **F** means the feature is fully supported, **N** is not yet supported and **P** is partly supported, **?** is not enough data collected.

Criteria	DAP	CardMeeting	AgilePlanner	MASE	Rally	VersionOne	XPPlanner	Scrumworks	GluWiki	APDT	Mingle	IBM Jazz
Creating, Editing and Deleting planning objects	F	F	F	F	F	F	F	F	F	F	F	F
Handle efforts estimates	F	N	F	F	F	F	F	F	N	N	F	F
Planning multiple iterations	F	P	F	F	F	F	F	F	P	P	F	F
Moving stories from one iteration to another	F	P	F	F	F	F	F	F	P	P	F	F
Authentication	N	F	F	F	F	F	F	F	F	N	N	N
Real-time updates of the plan	F	F	F	F	F	F	F	F	F	F	F	N
Visual characteristics for different types of stories	F	F	F	F	F	F	N	N	F	F	F	F
Integration with the development environment	F	N	N	N	N	N	N	N	N	N	N	F
Fluid transition between individual/collaborative work	P	P	F	P	P	P	P	P	P	F	P	P
Telepointers for pointing and gesturing	F	N	N	N	N	N	N	N	N	F	N	N
Real time information sharing	P	P	P	N	N	N	N	N	N	P	N	N
Change notification	P	P	N	P	P	P	?	P	N	P	P	P
Joining and leaving meetings	F	F	F	F	F	F	F	F	F	F	F	F
Fluid subgroup formation and dissolution	P	P	P	N	N	N	N	N	P	P	P	N
Simultaneous interaction	F	F	P	N	N	N	N	N	N	F	P	P

Fig. 13.5 Evaluation table of distributed agile planning tools

This evaluation table indicates that most requirements of agile project planning are supported by existing tools. Therefore, distributed agile teams can find an appropriate application for project management. However, the requirements for collaborative interactions are not yet or only partly fulfilled. Particularly, the key factors for synchronous interactions: *change notifications* and *telepointers* are rarely supported. The result of this tool analysis matched the results of our interviews with industrial developers. At present, several distributed agile teams are still using Microsoft NetMeeting and Skype for their agile planning meetings and no particular agile planning tools are utilized in the process.

We also conducted a survey to 54 project planning tools published on UserStories website [14]. We concentrate on knowing their application types (Browse, Native, Plugin), the license type (commercial, free), and major functions. The survey shows that:

- **application type:** 44 tools are designed for browser. 3 tools run as a native standalone system, and one tool is plugin to IDE. The remaining 6 tools provide multiple versions to support both plugin and native types.

- **license type:** 11 tools are free product, 22 are commercial product. The remaining 21 tools provide both commercial and free licenses.
- **supporting agile methods:** 15 tools are specifically designed for Scrum and 5 tools are for XP, while 10 tools provide general supports to both XP and Scrum development. The remaining 24 tools do not specify their supporting methods.
- **functionalities:** Only one tool supports synchronous planning, while others provide asynchronous planning features. The features include generating burn down charts, using board (Kanban, Scrum or story boards) to display planning data, strategy supports, timebox management, wiki publishing and agile management logs.
- **supporting multi-language:** Only one tool provides multi-language versions.

13.5 Practical Advice

In the 10 years of developing and working with agile planning tools along with the evaluation presented above, we present some advice for users and designers of distributed agile planning tools. In addition we discuss our assumptions on the future of distributed project planning tools.

13.5.1 Advice for Agile Planning Tool User

Distributed project planning tools are well developed for supporting asynchronous agile management. The diversity of application types, license types, and their functionalities, provide enough flexibility to choose appropriate tools for supporting specific requirements for asynchronous usage. By supporting, amongst others, Scrum, XP and Lean, agile project planning tools are not restricted to a specific agile methodology. However, when using existing agile planning tools, the following restrictions exist:

- For global agile development, multi-culture and languages are not supported. English is a dominating language for choosing distributed agile planning tools.
- Data exchange between different agile planning tools is problematic. Although existing agile planning tools conceptual support similar artifacts, it is difficult to exchange data between them. Migrating planning data between tools is time consuming and teams need to agree on using a single tool.
- The data exchanging issue also exists between communicating agile planning tools and some general project management applications. We observed some distributed teams using Microsoft Project to manage their development. However, only 2 out of 54 tools listed on UserStories website [14] are able to communicate with MS Project. Within the teams using MS Project and other agile planning tools, data exchanging are still conducted manually.

- Synchronous agile planning meetings are hardly maintained by existing tools. Despite some attempts, such as DAP and APDT were made at employing groupware and digital tabletop technologies to set up synchronous agile planning, none of the tools are fully commercialized and widely applied to industry. Some industrial agile teams are using non-agile specific groupware tools to address the problem. We found one team (distributed over two sites) using desktop sharing tools to set up a shared environment for agile planning meetings. While this is a pragmatic solution, it limits concurrent interactions and requires a single user at each site to interact with the planning workspace. Looking at long-term developments, we believe the support for synchronous agile planning meetings is becoming a trend with tool suppliers. We expect an increasing number of related tools to become available in the future.

13.5.2 Advice for Designers of Distributed Agile Planning Tools

Although most existing tools provide enough flexibility and functionality to support progress tracking, the collaborative interactivity needed for distributed agile planning meetings are insufficiently considered. As a result, the experience of industry in using agile planning tools can—and should—be enhanced. Moreover, the inability of sharing planning data between different tools will be problematic in the future. To better help distributed agile teams, we suggest that the following aspects will improve the usefulness and usability of existing agile planning tools.

- **Supporting synchronous interactions.** At present, synchronous interactions are not well supported by agile planning tools. Designing a practical synchronous interactive system needs to incorporate results from groupware research. Now that support for distributed project management has become more ubiquitous, suppliers need to distinguish their tools by properly supporting synchronous planning meetings. In addition, to enhance collaborative interactions, some advanced technologies need to be incorporated. It is highly possible that the next generation of agile planning tools might have to abandon the PC-projector displays and integrate with new interactive devices like digital tabletops. With the evolution of Web technology, Web browsers will soon support near real-time interactions and the accessibility of synchronous agile planning tools will be improved. The following factors should be considered when implementing synchronous tools for distributed agile planning:
 1. Verbal communication. Tools need to incorporate audio and/or video conferencing capabilities.
 2. A shared card-centered interface: A shared workspace is required for showing detailed aspects of cards, such as card colors, data on the card, and the card positions (considering teams often sort card to show their priorities).
 3. Showing the interactions of remote participants. One of the goals of distributed planning tools is to enhance the collaboration across multiple sites. However,

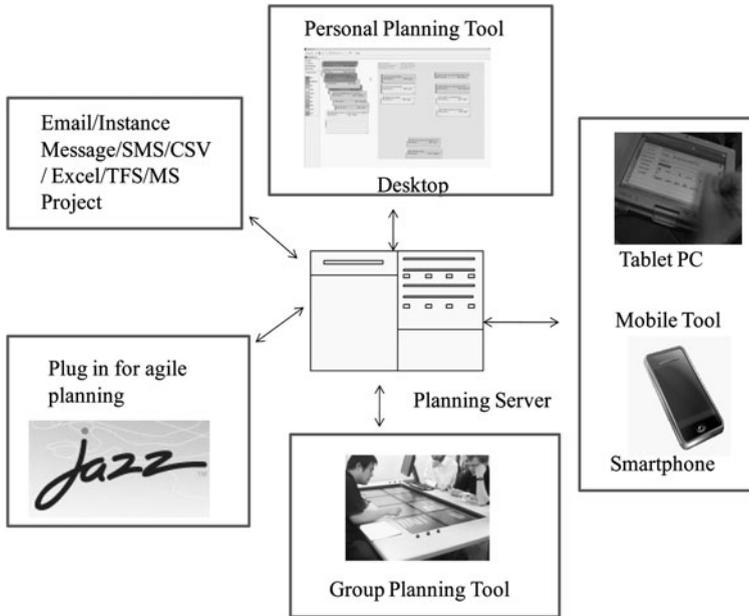


Fig. 13.6 Ubiquitous project planning model

existing tools do not yet show the remote participants' interactions. Telepointers are an appropriate approach to show distributed user interactions. By using remote mouse pointers to monitor the users' interactions (clicking, dragging), the telepointer will show who is interacting with the workspace, and what they are doing. Telepointers also help with identifying the focus of discussion across remote sites. Digital tabletops can show arm shadows (the shadows of users' arms on the table screens) [11]. Thus, the users can not only see hand movements, but also recognize the hand or arm gestures.

In designing synchronous agile planning tools, one needs to combine and implement the above factors in an appropriate manner. Admittedly, some advanced features (such as showing telepointer, or arm gestures) might be restricted by the hardware or software platforms. However, maintaining a card-centered, real-time distributed workspace as well as an audio/video communication are required to support distributed agile planning.

- Ubiquitous project planning.** Agile project planning often includes multiple roles, such as developers and managers. The diversity of participant raised several types of requirements for having access to agile planning. For example, project managers would like to read the project process reports or burn down charts on their mobile devices. Software developers highlight the use of project planning plugin for their IDE. Meanwhile, both of them would like to have a convenient environment when sitting together to communicate with another distributed sub-teams. Generally, everyone wants to view the project progress from their personal

computing environment. In Fig. 13.6 we proposed a model to serve agile planning participants at different environments. Parts of this model, such as plugins and personal planning tools have been implemented. Other components, such as the tabletop based planning tools are still being developed or evaluated. However, a challenging issue for implementing the model is how to exchange data between the various tools. To solve this issue, the following requirement becomes necessary.

- **Exchanging planning data among different tools.** Although agile project planning tools are essentially representing very similar information, none of them can easily exchange planning data with each other. Current tools are closed and create supplier lock in. To solve this issue, we explored the feasibility of exchanging planning data among different tools. We found most agile planning tools were based on similar conceptual models. A simple translator should be able to bridge the terminology differences among existing tools. For example, we modified a DAP server and added a gateway to translate the planning data from DAP to IBM Jazz and from DAP to the Rally tool. We believe that the integration of different tools will become increasingly important as multiple teams will have to collaborate when agile projects are scaled up. Moreover, card display, editing and management is not enough to support the complete process of agile development. Bug tracing, version control and test automation also play a significant role. Thoughtworks Studio [13] integrates card-based project planning tools (Mingle) with its own release management (version control) products (Cruise), and automation testing platform (Twist). The similar idea is also implemented by the new Microsoft VSTS 2010. Admittedly, it is not easy for most developers, particularly those individual developers developing a complete software package that covers from card planning to release management. However, in designing an agile planning tool, it is beneficial to reserve some extension points (such as data structures that keep the path of one or multiple source code files) to allow source control tools or testing platforms binding their code segments, version updates or testing cases with digital index cards in the agile planning tool.

13.6 Conclusions

Project management tools for distributed agile teams are currently widely available. They have shown some benefit to supporting project management and knowledge sharing. However, our analysis of existing tools shows that nearly all of them focus on asynchronous features such as supporting progress tracking and card management. The next major step in distributed agile planning tool development will require: supporting synchronous project planning meetings, setting up ubiquitous project planning environments and enabling data exchange between different agile tools and/or with non-agile project planning. These enhancements still require substantial research and development combining agile software development expertise with knowledge about computer supported work and groupware.

References

1. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile software development methods*. Lspoo: VTT Publications. p. 112.
2. Chau, T., & Maurer, F. (2004). Tool support for inter-team learning in agile software organizations. In *Proceeding of the workshop on learning software organization* (pp. 20–21), Banff, Canada, June 2004.
3. Chau, T., & Maurer, F. (2005). A case study of wiki-based experience repository at a medium-sized software company. In *Proceedings of the 3rd international conference on knowledge capture*, Banff, Canada.
4. Dyck, J., Gutwin, C., Subramanianand, S., & Fedak, C. (2004). High-performance telepointers. In *Proceedings of the 2004 ACM conference on computer supported cooperative work* (pp. 6–10), Chicago, US, November 2004. New York: ACM.
5. Ellis, C. A., Gibbs, S. J., & Rein, G. (1991). Groupware: Some issues and experiences. *Communications of the ACM*, 34(1), 39–58.
6. Grudin, J. (1994). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1), 92–105.
7. Jazz Overview (2009). IBM Jazz. <http://jazz.net/>. Cited 13 Nov. 2009.
8. Larman, C. (2004). *Agile & iterative development—a managers’s guide*. Boston: Addison-Wesley (pp. 25–34).
9. Morgan, R., & Maurer, F. (2008). An observational study of a distributed card based planning environment. In *Proceeding of the 9th international conference on agile processes and eXtreme programming in software engineering* (pp. 10–14), Limerick, Ireland, June 2008. Berlin: Springer.
10. Rally Tool (2010). http://www.rallydev.com/agile_products/agile_planning/. Cited 13 Jan. 2010.
11. Robinson, P., & Tuddenham, P. (2007). Distributed tabletops: Supporting remote and mixed-presence tabletop collaboration. In *Proceeding of 2nd workshop on horizontal interactive human-computer systems* (pp. 10–12). Newport, US, October 2007.
12. Scott, S. D., & Carpendale, S. (2006). Interacting with digital tabletops. *IEEE Computer Graphics & Applications*, 26(5), 24–27.
13. Thoughtworks studio (2010). <http://www.thoughtworks-studios.com/>. Cited 13 Jan. 2010.
14. User Stories (2009). <http://www.userstories.com/products>. Cited 13 Nov. 2009.
15. Wang, X., & Maurer, F. (2008). Tabletop AgilePlanner: A tabletop-based project planning tool for agile software development teams. In *Proceeding of the 3rd symposium of tabletop and interactive surface*, Amsterdam, 1–3 October 2008.

Chapter 14

Combining Agile and Traditional: Customer Communication in Distributed Environment

Mikko Korkala, Minna Pikkarainen,
and Kieran Conboy

Abstract Distributed development is a radically increasing phenomenon in modern software development environments. At the same time, traditional and agile methodologies and combinations of those are being used in the industry. Agile approaches place a large emphasis on customer communication. However, existing knowledge on customer communication in distributed agile development seems to be lacking. In order to shed light on this topic and provide practical guidelines for companies in distributed agile environments, a qualitative case study was conducted in a large globally distributed software company. The key finding was that it might be difficult for an agile organization to get relevant information from a traditional type of customer organization, even though the customer communication was indicated to be active and utilized via multiple different communication media. Several challenges discussed in this paper referred to “information blackout” indicating the importance of an environment fostering meaningful communication. In order to evaluate if this environment can be created a set of guidelines is proposed.

14.1 Introduction

The steadily increased popularity of agile development methods has resulted in agile growing out of its infancy and entering from small-scale collocated development projects into a world of globally distributed large software enterprises with all their harsh realities. Distributed environment is already challenging in a world of tradi-

M. Korkala (✉) · M. Pikkarainen
VTT Technical Research Centre of Finland, P.O. Box 1100, 90571 Oulu, Finland
e-mail: Mikko.Korkala@vtt.fi

M. Pikkarainen
e-mail: Minna.Pikkarainen@vtt.fi

K. Conboy
National University of Ireland Galway, Newcastle Rd., Galway, Ireland
e-mail: kieran.conboy@nuiagalway.ie

tional software development.¹ In fact, it seems to be so complex that either comprehensive understanding on the problem domain, or the potential success factors are not yet thoroughly known [2, 3].

Agile development emphasizes intense communication between the stakeholders involved in the project. This tenet has been stretched into extremes by Extreme Programming (XP) [4] which proposed an *on-site customer* practice which promoted full-time customer participation, thus enabling constant face-to-face communication and immediate feedback. In reality, having an on-site customer can be very difficult and solutions for intense communication have to be found out through other means. Naturally, several solutions to mitigate the problem have been introduced. Audio-visual communication media approach the problem from the technical side, while more general level proposals have been introduced e.g. by Layman et al. [5], Ramesh et al. [6] and Boehm & Turner [1]. The use of different communication media and applying different coping strategies may relief the problems caused by significant geographical distances which hinders the possibilities of face-to-face communication between the participants. Naturally, cultural aspects and the distance created by them should be taken into account.

At the moment, scientific literature provides only few specific studies listing the challenges and solutions for customer communication in a distributed agile development context. In order to find out these challenges and provide practical solutions to companies, we conducted a case study which sheds some light on this less studied field. In addition, we propose some guidelines in order to ensure if the development environment itself can foster meaningful communication between the distributed partners.

14.2 Customer Communication in Distributed Agile Development

Communication plays a significant role in the success of all software development, regardless of the underlying development approach, but it is particularly emphasized in agile development as being one of the corner stones of the approach. Communication however is a difficult task requiring a common understanding on the topics being discussed [7]. Taking the challenging nature of communication into account, it is not perhaps difficult to agree that communication and coordination breakdowns seem to be much more than just occasional incidences of minor significance. Instead, they seem to be a commonplace phenomenon. Despite the fact that efficient

¹Traditional development is synonymous with plan-driven development. Plan-driven software development is an engineering approach in which the software is developed following specific processes, commencing at the requirements gathering stage and ending with the final code [1]. There are several methodological approaches and models describing how to develop plan-driven software [1], the best known probably being the ‘waterfall’ model in which all the development phases are implemented, at least twice at stages after one another to be able to produce the working software.

Table 14.1 Recommendations by Layman et al. [5]

Proposed recommendations

-
- (a) Define a person to play the role of the customer up front. This individual must be able to make conclusive decisions on project functionality and scope, must be readily accessible, and must have a vested interest in the project.
- (b) When the project management and development teams are separated, create a role within the XP team whose purpose is to work closely with both development and project management teams on a daily basis, preferably someone who speaks all the languages involved.
- (c) When face-to-face, synchronous communication is infeasible, use an email listserv to increase the chance of a response and encourage prompt, useful, and conclusive responses to emails.
- (d) Use globally-available project management tools to record and monitor the project status on a daily basis.
-

and interactive customer communication is a central tenet in agile development, little is known about its actual impacts on development. Some work has however been conducted focusing e.g. on the quality of the software [8] indicating a relationship between the increasing of defects and customer involvement.

Naturally, several approaches for mitigating the risks distribution creates for agile communication have been proposed. Tools capable of mediating audio-visual communication are a natural choice, but also more general level solutions have been suggested. E.g. Layman et al. [5] have proposed a set of general guidelines that aim to create a communication-rich environment for distributed agile development. These guidelines are presented in Table 14.1.

In addition, Ramesh et al. [6] have identified five different key areas that are challenging in distributed development. One of these challenge areas is related to communication and is referred as *Communication need vs. communication impedance*.² According to [6], the balance between formal non-rich communication channels and informal rich media should be found in distributed agile development. Gottesdiener [9] has defined a framework for holding requirements workshops with various time and place combinations. This framework can be applied to distributed agile development, thus indicating that the stakeholders can select a proper communication channel from considerably wide collection of different media. In this work, time can be either synchronous with every participant present at the same time or asynchronous. Place can be either co-located or distributed when participants are at different locations [9]. As can be suggested based on what was depicted above, there is a plethora of different communication media and proposals available for improving the success of communication and ultimately, the development project.

Theory suggests that attention should be paid on selecting the appropriate communication medium for different tasks. Despite the criticism, Media Richness The-

²Originally, impedance is a quantity related to electricity. Since the term “communication impedance” is not established in SW literature, we believe that the term deals with similar understanding of the message by all the parties involved in the context which it is used by Ramesh et al. [6].

Bureaucratic organization is a difficult environment for agile to succeed. Bureaucracy is defined as an organizational form based on specified roles, positions and strict rules. Thus, it has a direct impact to the efficiency of customer communication in an organization by possibly restricting the communication. Bureaucratic culture can be seen as hierarchical, procedural, regulated, established, structured, cautious and power oriented. The culture of a group can be defined as: A pattern of shared basic assumptions that the group learned as it solved its problems of external adaptation and internal integration, that has worked well enough to be considered valid and therefore, to be taught to new members as the correct way to perceive, think, and feel in relation to those problems. In addition, *blame culture* is prevalent in bureaucratic and hierarchical companies. In a study reported in [12] the level of culpability went hand in hand with the status of the representatives of upper management. Despite the efforts, the culture of blame remained prevalent in the company. Even though these challenges are not discussed specifically on the distributed context, it can be argued that they are prevalent also within companies working in such fashion.

Another factor affecting to the customer communication in the agile company are **individual attitudes** such as fear of change and either specialized or outdated skills to fears of losing control especially within the management. Individual attitudes may hinder agile communication in many ways; for example, fear of change may result in resistance against informal and active customer communication, if formal and less personal communication methods have been used as a standard and normative way to manage customer communications.

Naturally, **geographical distance** creates challenges in using agile for distributed environment. Distances between the different parties can be vast and especially distance across time zones is significant factor separating the collaborating stakeholders. In fact, temporal distance seems to be more dominating factor than plain physical distance [13]. However, if face-to-face communication is not possible, the natural approach in agile context would be to pursue for as efficient communication channels as possible. As previously explained, there are several different solutions available for communicating with distant stakeholders.

14.3 Findings

This section presents the findings of the case study including description of the case context, characterization of the analysed organizational units and evaluation of the customer communication richness, challenges and solutions.

14.3.1 Case Context

Our case study focuses on analysing customer communication in a globally distributed environment comprised of two organization units implementing the same

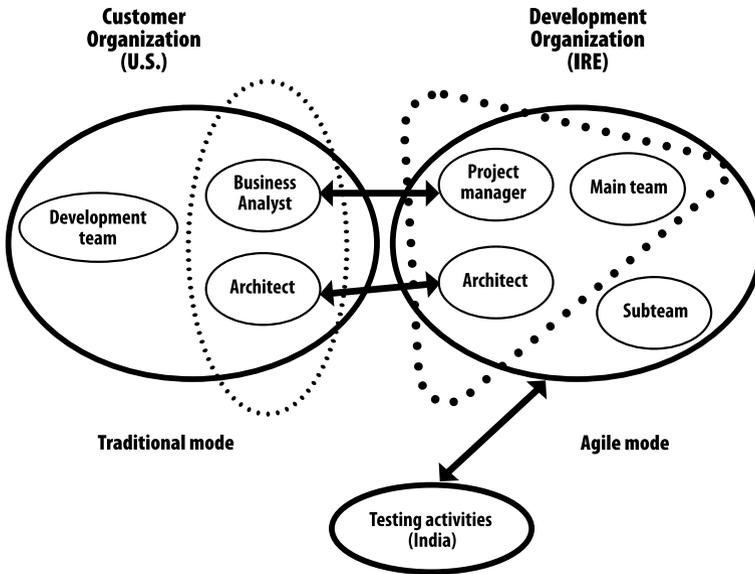


Fig. 14.2 The project organization and customer communication environment of the case project

system within a large globally distributed software intensive company. Agile methodologies had been used in the organization for over two years at the time of the data collection. However, this project was the first agile effort for many of its participants.

A team based in the U.S. was implementing the front-end functionality of the product, while the implementation of the back-end was allocated to two teams based in Ireland. Additionally, the project had a quality engineer in India. The Business Analyst and the Architect of the U.S. organization assumed the roles of the customer representatives (i.e. the main sources of information for their Irish counterparts) and provided requirements for the Irish organization. These requirements were defined and agreed up-front in a traditional manner. The project environment along with the identified customer communication links between the U.S. and Ireland are depicted in Fig. 14.2. In addition, the elements inside the triangle depict the interviewed stakeholders and further illustrate the unit of analysis within this study. Similarly, the customer representatives of the U.S. unit are circled.

The project manager was responsible for management topics (such as agreeing on schedule, budget and resources) while the architects communicated more about technical aspects such as architecture and its technical implementation between the different development organizations.

The overall relationship between the Irish teams, the US team and the Indian quality engineer was studied in order to understand the context of the case study. The Irish main team consisted of experienced developers who were responsible for difficult tasks. The subteam was composed of junior developers who implemented

simpler items allocated by the Irish Architect. The two teams were collocated in same facilities and the Irish architect was the customer for the subteam, providing and clarifying them the requirements that should be implemented. Since the teams were collocated, the Irish architect—Irish subteam relationship is similar to onsite customer relationship. The relationship between the Irish teams can be seen as a distributed project with collocated teams. The Irish subteam had their own customer representative who steered their development and prioritized their requirements. Altogether, it was found that the collaboration between the collocated Irish teams was fluent, since significant challenges related to this co-operation were not mentioned in the interviews. It is quite safe to argue that collocation played an important role in this fluent collaboration. However, from a larger perspective this project can be seen as a distributed project with distributed teams since the organizations were implementing a single product. This latter viewpoint is in the focus of this paper.

As mentioned, the Irish organization also utilized testing services located in India. Based on the interviews, this collaboration worked without any significant problems. Since the U.S. organization was working as a customer for the Irish branch, the case could be observed also from outsourcing and offshoring perspectives. Furthermore, since the Irish branch had offshored its testing services to India, there is a strong indication that they might have been working as a “bridge”, working both as a customer for the testing organization in India and as a vendor for the U.S. unit. The “Irish Bridge” concept is described by Holmström et al. [14].

In this work, distributed development angle is selected due to the fact that the participants and customer representatives belonging to a same company were implementing equally important parts of the same product in geographically distributed locations. However, customer-vendor relationship is critical in the success or failure also in offshored arrangements [14]. Thus the presented challenges can be valid and proposed solutions could be well adapted also in offshoring context. Since we were able to interview the representatives of the Irish branch only, we observe the customer communication challenges within the project from the viewpoint of the Irish branch. The lack of data from the U.S. organization is a recognized limitation to this work.

14.3.2 The Use of Agile Methodologies in the Case Project

At the time of the interview data collection, there was a distinct variance between the level of agility in the Irish and US units of the case company. Table 14.2 below provides an overview to the units and their characteristics following the categorization by Boehm and Turner [1] explained earlier in Table 14.1. Table 14.2 indicates that there were hardly any agile elements present in the U.S. organization.

The agile practices used in the Irish organization were a combination of practices from Extreme Programming and Scrum. The usage of practices is described in Table 14.3. Very often usage of a practice is not binary, and so we have divided the practices in this case into those that are used routinely, occasionally, and not used at all.

Table 14.2 Characteristics of the case organizations based on Boehm and Turner [1]

	Irish organization	U.S. organization
Customer Relations	Dedicated on-site customer representative (Architect) for the Irish subteam. Increments prioritized internally.	As needed customer interactions; Active communication with the Irish unit.
Planning and Control	Internalized iterative planning; Contents of the iteration planned and estimated.	Documented plans; Steering based on milestones. No interaction with the Irish unit considering their internal planning and releases.
Communication	Tacit interpersonal knowledge; face-to-face communication within the Irish organization.	Explicit documented knowledge; steering based on documentation. However, active communication between the Irish unit using different media.
<i>Technical</i>		
Requirements	Formalized project, capability, interface, quality, foreseeable evolution requirements	
Development	Simple design; short increments: Internal design process simplified from the previous approach. Four week iterations	Extensive design; longer increments; Up-front defined plans and designs, very long increments (over 6 months)
Testing	Documented test plans and procedures	
<i>Personnel</i>		
Customers	U.S. organizations stakeholders considered customers for the Irish branch. Irish architect working as an onsite customer for the Irish subteam. Customers were not analysed based on CRACK (Collaborative, Representative, Authorized, Committed, Knowledgeable) criteria.	
Developers	Not analysed in this study. Not in the focus of the study.	Not analysed in this study. Not in the focus of the study.
Culture	Comfort and empowerment via many degrees of freedom (thriving on chaos): Agile approach used.	Comfort and empowerment via framework of policies and procedures (thriving on order): Bureaucratic environment.

14.3.3 The Use of Customer Communication Media

In this section, both the usage of different communication media is described. As can be seen from Fig. 14.3, the usage of different customer communication channels was active and the communication itself was conducted through several different media varying in their effectiveness. The amount of use of different channels is indicated by (+) signs.

Table 14.3 The use of agile practices in the Irish organization

Combined list of agile practices	Extent of use in Irish organization
Pair Programming	Not Used
Testing	Used Occasionally
Metaphor	Not Used
Collective Code Ownership	Used Routinely
Refactoring	Used Routinely
Coding Standards	Used Routinely
Simple Design	Used Routinely
40 hour week	Used Occasionally
On Site Customer	Not used/Used Routinely ^a
Sprints	Used Routinely
Sprint Planning	Used Routinely
Architecture	Used Routinely
Sprint Review	Used Routinely
Post Game Sessions	Used Occasionally
Daily Meetings	Used Routinely

^aThe Irish main team prepared requirements for the sub-team and steered them. The Irish architect worked as an On-Site customer for the Irish sub-team

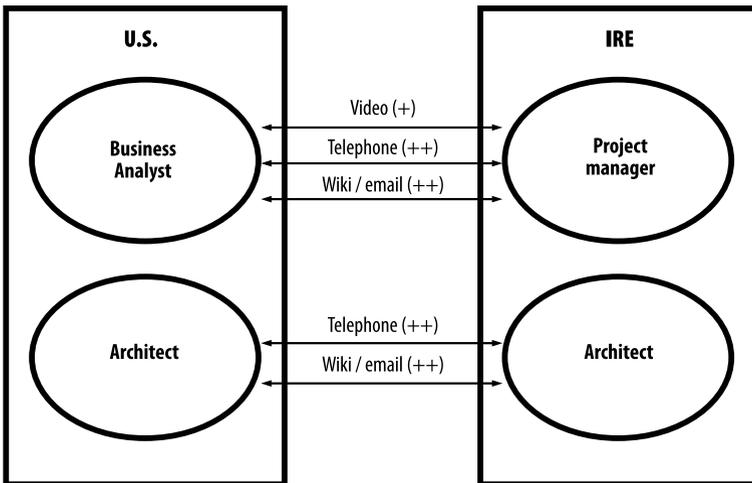


Fig. 14.3 The usage of different communication media in the case project

Even though there was not a single communication media that was used more actively than others, the interviewees considered the project wiki to be the central medium largely due to the distributed nature of the effort. Face-to-face communica-

Table 14.4 Customer communication media and their usage in the project

Communication medium	Usage
Face-to-face	Utilized in the beginning of the project for a couple of weeks by onsite visit in the U.S. Not utilized later. No findings from the time of the study.
Video conferencing	PM: Higher level decisions, analysis discussions. Topics focused on dates, etc. Video conferences used occasionally, the usage decreased since the projects inception. Videoconferencing not used by the Irish architect.
Teleconferencing	Teleconferencing was used extensively. PM: Weekly customer meetings, weekly program meeting, steering meeting in every two weeks. Solving urgent (management) issues when needed. Architect (IRE): Scheduled conference call twice a week, informal communication in daily basis. A separate meeting had to be organized a couple of days in advance in order to solve problems. No instant feedback.
Wiki/Email	Used extensively by everyone involved in the project. Wiki: focus on technical topics. Documents stored in the wiki. Meeting minutes and high level planning issues documented in the wiki. Wiki was the main Ireland–U.S. customer communication channel during the project. The amount of information in the wiki was extensive and the wiki had become very difficult to navigate. Email: Email was the main decision making medium between the Irish architect and the customer (U.S. architect. Decisions documented in the wiki).

tion is missing from the picture which describes the communication channels used during the interviews. However, face-to-face communication was used between the U.S. and Irish organizations in the traditional type of specification phase during the first three months of the project. In this stage, the Irish analyst spent “couple of weeks onsite” with the U.S. representatives when the initial requirements specification workshops were held. Developers from the U.S. organization and developers from the Irish branch had never met personally. In addition, the Irish developers had never seen the U.S. customer representatives.

Table 14.4 discusses the usage of these media in more detail. In addition, the purposes in which the media were used are described. In this case, **PM** refers to the Irish project manager.

Overall, the customer communication on the project was active. Both rich and non-rich media were used and communication took place in daily basis, excluding the usage of videoconferencing. Videoconferencing was used in the Irish branch only by the project manager and the topics themselves focused on very high level issues such as explained earlier. On the other hand, the architects communicated about more technical aspects.

The customer communication was seen sufficient by the Irish project manager and the Irish architect. The reasons for this varied, as indicated by the following comments: “*we are communicating pretty much on a daily basis. And, if there is a particular issue, there isn’t a requirement to wait for a forum, you just pick up the phone.*” (Project Manager): “*It’s sufficient for the level we’re at. Again, if the customer was more focused on delivery, we’d probably need more, but because they’re not that focused on it.*” (Irish architect). The architects comment indicate the lack of customer support which is one of the challenges discussed in the next section.

14.3.4 Identified Customer Communication Challenges

In this section, the identified customer communication challenges are discussed. In addition, also other interesting factors that we found to have contribution to the presented challenges are discussed.

Detached Customer Despite active communication between the Irish and U.S. branches, the customer relationship itself left a lot to be desired. This uninvolvement was considered as a significant downside. The detached nature of the customer organization and its representatives manifested itself most clearly while discussing the requirements that were supposed to be implemented by the Irish organization. As mentioned, the fixed requirements provided to the Irish branch were defined up-front. However, the use of agile was used as an “excuse” to make the requirements definition more loosely than before while an entirely traditional development approach was being used. The level in which the requirements were now defined would have had required active agile type of interactive planning and active communication between the U.S. and Irish branches in order to clarify their contents in more detail while the development progressed. This was however not the case. The customer representatives communicated with the Irish unit about high level aspects. The Irish organization defined their requirements in more detail by themselves in their internal iteration planning meetings without any customer support. As the following quote indicates, the Irish branch was not happy with this approach. “*They’re not great customers, because they can just keep talking at a very high level without actually giving detailed requirements*” (The Irish Architect).

What further complicated the situation was that the Irish development teams did not have any previous knowledge on the domain. This combined with the lack of support from the U.S. unit can be well considered hazardous, since the Irish organization had to rely on their “best educated guesses” on how the functionalities should work. Agile promotes continuous integration which can be considered as a tool for verifying whether the software does what it should be doing. Since the front- and back-end functionalities had not been integrated at any point during the development process, there was not a mechanism to ensure that the requirements were implemented the way they should have been.

In addition, detachment of roles was visible also internally in the Irish organization. The project manager, as explained, was focusing on managing higher level

aspects of the project and was not concerned much on the implementation or the architecture of the product. On the other hand, the U.S. and Irish architects were communication technical viewpoints. These topics however did not include software's functional requirements but focused more on technical details, such as deciding what database drivers should be used.

The central communication challenge emerging from customers' detachment in this case is the *meaningfulness of communication*. As it was found, the communication was very active and involved several different communication channels varying in their level of effectiveness. The contents of the communication however did not serve the purpose well from the Irish organization's point of view, since the teams did not receive any information about the details of the requirements. Overall, a lot of communication took place, but it did not directly support the development as it should have been doing.

Organizational Environment Bureaucratic culture can be seen as hierarchical, procedural, regulated, established, structured, cautious and power oriented. Indeed, several characteristics of a bureaucratic organization were prevalent. The customer organization was reluctant to openly share information with the contractor, even though they were implementing the same system. The following comment indicates the resistance of openly sharing the relevant information i.e. cautiousness: *"We're working for a part of the organization which has typically worked by themselves, and to join them is a very political issue... and also because it's financed, and it's sensitive data, they always try to hide it"*. Thus, it seems that the reasons for information hiding emerged from the politics applied by the customer organization. In this particular case, the protective nature of the organization can be traced directly to cautiousness of a bureaucratic environment.

Naturally, unfamiliarity of agile processes in itself can create uncertainty and misunderstandings but again one characteristics of a bureaucratic organization emerged related to the "status-blame" relationship. There were indications that upper management feared that if the project should fail, they could lose their jobs: *"at the high level there's some worry around about are they losing their jobs"*. (Irish architect about the consequences of failure). There is actually nothing new with this finding since similar findings related to "status-blame" relationship has been made previously in bureaucratic environments.

Another finding that supports hierarchical environment, which again is a characteristic of a bureaucratic company, was the indication of tightly defined responsibilities. The employees of the organization "did not cross boundaries" and focused only on their specific expertise areas. Clear division of responsibilities is counter-intuitive to agile which promotes cross-functionality of the teams. This aims to increase the knowledge on the software being implemented in the project among the team members. Since the team members are not limited to work on their corresponding expertise areas alone, they should have more holistic view to the product being implemented. In this case, the Irish project manager was supporting the work of the Irish teams more from the management side without involving into technical development work. To further illustrate the tight division of roles, the Irish developers

were deliberately excluded from communicating directly with the customers. The reason behind this decision was to ensure that the developers will have the maximum amount of time available for product development.

Tightly defined roles can create challenges for communication. The lack of first hand experience on a certain topic might result into misunderstandings and misinterpretation of data. In addition, if information has to pass through multiple persons there is a chance of information distortion, which means that some of the information is lost and some of the contents mutated [15]. The longer the communication chain is, the more the information distorts.

Altogether, organization's environment in this case proposed a significant challenge to projects execution, namely in a form of deliberate *information hiding*. In addition to the lack of meaningful information considering the details of the requirements the Irish organization were implementing, they did not had any access to the code implemented in the U.S. despite it was a part of the same product. The access to this program code could have provided more information on the functionality of the requirements allocated for the Irish branch.

Differences Between Traditional and Agile Approaches The U.S. organization worked following a milestone oriented approach, which translated to communication also. In fact, the customers did not want to be involved in "agile communication", namely participating to the Irish teams Sprint Plannings and Sprint Reviews. Instead, they wanted to take a more traditional approach and communicate higher level topics, as described earlier. Indeed, the U.S. customers communicated actively with their Irish counterparts but remained more distant than expected from the agile perspective, since development level issues were not communicated.

This finding indicates the difficult situation in which agile teams working with traditional teams might encounter; they are not supported they way they should be. This in turn might have a serious negative impact on the project results, which in agile context are tried to be achieved through active communication and feedback. The traditional communication approach taken by the U.S. organization also suggests that agility of the Irish organization did not have any impact on customer communication. Furthermore, Irish unit's agility did not have any impact on the development approach taken in the U.S. These two organizations were very detached, even though they were working on a same product.

Lack of Trust Even though indications of bureaucratic organizational characteristics together with detached customer and the differences between agile and traditional approaches seemed to be the cause behind ineffective communication, the issues related to mutual trust were also identified. The following comment made by the project manager on the sense of belonging to the same team indicates that mutual trust had not been evolved: "*Even for trust, on both sides to build, whereby it would be totally frank and open, it's still early days*". (Project manager)

Similar situation has been documented in [13], explaining that two different team without previous experience working together lacked trust on each other. This in turn, hindered the promotion of their effort to their project sponsors [13]. In fact,

building mutual trust seems to be the key for solving different technical issues efficiently. Thus, lack of trust might have contributed to other challenges mentioned.

14.4 Discussion and Lessons Learned

The key lesson learned from this case can be condensed to a single sentence: *It does not matter how much you communicate, it is what you communicate*. The information communicated should be meaningful and serve the purpose. Agile development relies heavily on active and rich communication in order to solve the problems and steering of development and elaboration of software requirements, just to name a few examples. Even though the different organizations communicated actively, it was clear that the contents of customer communication in this case were only related to the higher level management and technical aspects and did not serve the needs of the unit working in agile mode. It can be said that the Irish unit suffered from *information blackout* since the U.S. customer unit did not provide them necessary information, feedback and steering considering the requirements the Irish unit were implementing. This kind of information is essential since agile development aims to provide value to the customer in a form of working software as soon as possible. In this case the progress of the project was tracked against agreed milestones and dates instead of customer value.

There is quite likely overlapping and cause and effect relationships present between the challenges discussed in this paper. For example, traditional development was customary in within the U.S. customer unit and they stayed with this approach also in this project. Customer relationship is far more distant in traditional development than in agile approaches. In this case the organizational differences, the clash of traditional and agile, might be one of the explaining factors of the customers' detachment. On the other hand, organizational environment might have affected to this, as well the as the lack of trust between the participants. However, these relationships are not discussed any further in this work. However, analysing these relationships would be an interesting topic for a further research.

There is existing work on how to mitigate the risks caused by lack of customer involvement and clashes between traditional and agile development, but few notes on these viewpoints should be given also in this work.

Detached Customer It is essential to understand that the customer is a vital part of agile development. In agile development, the customer is more than a distant entity providing a set of requirements and then disappearing. On the contrary, customer is an integral part of the team, ideally available for interactive discussions when ever needed. Thus it is quite natural that the role of agile customer is often considered to be more demanding than in traditional development and sometimes the customers might be even unwilling to participate actively to the development process. Anyone considering the role of agile customer should realize the expectations of the role and accept those responsibilities. However, one cannot always have the luxury of committed and active customer. In these situations, customer proxies could be used.

However, there is a downside in this approach due to the possibility of information distortion, since the proxy might have to negotiate with the real customer and then translate the customer needs to the development organization. It is always better to use as short chain of communication as possible.

Considering the *Differences between traditional and agile approaches*, perhaps the most common propositions are to carefully assess the best possible development approach and find a balance between these two approaches. For example, Boehm & Turner [1] provide guidelines for the first alternative, while Ramesh et al. [6] propose to find a balance between these two different development methodologies. *Trust* on the other hand is something that is evolving during time. E.g. paying visits to other sites may help establish trust between the distributed partners.

Based on the key learning from this work, we propose the following guidelines based on our insights to identify whether it is possible to have an environment for meaningful customer communication. Without this environment, development might become very challenging and rendering the potential benefits of agile redundant. If this environment is not available, the use of agile practices should be re-evaluated. These guidelines are presented in a form of questions that should provide the necessary information:

- ***What kind of information do we need from the other party?***
This information can be for example software requirements, access to the code implemented by other organizations, etc.
- ***Who will be providing this information?***
This refers to the customers and other relevant stakeholders who provide the information relevant to successful completion of the work. These should be identified.
- ***Are we able to get this information when it is needed?***
Agile development is cyclical which means that some of the information, for example customer support during the Sprint Planning, is required at agreed intervals. On the other hand, the customer should provide information critical to the development as soon as possible after it has been requested. As an example, the discussed and agreed requirements might still include ambiguous elements which should be clarified as soon as possible. This question aims to clarify if necessary information can be obtained in a timely manner and if the customers are able to provide it when it is needed.
- ***Are the sources of information committed to provide the information when agreed?***
In addition to the information itself, the source of the information is equally important. As explained earlier, the role of the agile customer is more demanding than in traditional development. It is essential that the stakeholders providing the information are willing and able to do so.
- ***Is there something that prevents us from getting this information?***
As described in this paper, organizational characteristics may prevent the access to information from other stakeholders. The possible obstacles should be identified and solutions to mitigate the problems developed.

This check-list is by no means conclusive, but it could be used as a starting point. In addition, this list does not discuss what communication media should be used. Ex-

isting work promotes the usage of rich communication media, but since distributed development can cross several time-zones, using interactive media can be extremely difficult due to temporal distance. Thus, the decisions on what communication media to use is left for the collaborating parties based on their needs and capabilities. As described, there is a wide collection of different communication solutions available.

References

1. Boehm, B. W., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed*.
2. Komi-Sirviö, S., & Tihinen, M. (2005). Lessons learned by participants of distributed software development. *Knowledge and Process Management*, 12, 108–122.
3. Lee, G., DeLone, W., & Espinosa, J. A. (2006). Ambidextrous coping strategies in globally distributed software development projects. *Communications of the ACM*, 49(10), 35–40.
4. Beck, K. (2000). *Extreme programming explained: Embrace change*. Upper Saddle River: Addison-Wesley.
5. Layman, L., Williams, L., Damian, D., & Bures, H. (2006). Essential communication practices for Extreme Programming in a global software development team. *Information and Software Technology*, 48, 781–794.
6. Ramesh, B., Cao, L., Mohan, K., & Xu, P. (2006). Can distributed software development be agile? *Communications of the ACM*, 49(10), 41–46.
7. Cockburn, A. (2002). *Agile software development*. Indianapolis: Addison-Wesley.
8. Korkala, M., Abrahamsson, P., & Kyllönen, P. (2006). A case study on the impact of customer communication on defects in agile software development. In *AGILE 2006* (pp. 76–86).
9. Gottesdiener, E. (2002). *Requirements by collaboration*. Upper Saddle River: Addison-Wesley.
10. Daft, R. L., & Lengel, R. J. (1986). Organizational information requirements, media richness and structural design. *Management Science*, 32, 554–571.
11. Daft, R. L., Lengel, R., & Trevino, L. K. (1987). Message equivocality, media selection, and manager performance: Implications for information support systems. *Management Information Systems Quarterly*, 11, 355–366.
12. Berger, H. (2007). Agile development in a bureaucratic arena—A case study experience. *International Journal of Information and Management Sciences*, 27(6), 386–396.
13. Treinen, J. J., & Miller-Frost, S. L. (2006). Following the sun: Case studies in global software development. *IBM Systems Journal*, 45(4), 773–784.
14. Holmström, H., O. Conchuir, E., Åkerfalk, P. J., & Fitzgerald, B. *The Irish bridge: A two-sided perspective on the customer-vendor relationship in offshore sourcing*. Presented at 29th Information Systems Research Seminar in Scandinavia, Helsingoer.
15. Melnik, G., & Maurer, F. (2004). Direct verbal communication as a catalyst of agile knowledge sharing. In *AGILE 2004* (pp. 21–31).

Chapter 15

Coordination Between Global Agile Teams: From Process to Architecture

Jan Bosch and Petra Bosch-Sijtsema

Abstract Traditional process-centric software development has served software-intensive companies well for decades. During recent years, however, the trends of increased adoption of software product lines, software ecosystems and in particular global software engineering have led to unmanageable complexity and unacceptable overhead. In this paper we present research performed at three global companies in which we studied the relation between large-scale and agile approaches to software development as well as current problems. In addition, by integrating the best practices adopted at the case study companies, we present an alternative approach: architecture-centric software engineering. This approach largely removes inter-team dependencies and provides much higher efficiency and productivity in global software development contexts.

15.1 Introduction

For four decades now, software engineering continues to be a fascinating field. With Moore's law, the network law and the storage law doubling capacity every 18, 12 and 9 months, respectively, the size of the software systems on top of the hardware and communication networks is growing at similar rates. One can find examples of this within large Internet companies, e.g. around search engines, the IT systems supporting Fortune 100 companies and in the software ecosystems surrounding large platforms, ranging from PC operating systems to mobile devices. As a consequence, the

P. Bosch-Sijtsema is visiting scholar at Stanford University, Stanford, CA, USA.

J. Bosch (✉)
Intuit, Mountain View, CA, USA
e-mail: Jan@JanBosch.com

P. Bosch-Sijtsema
Aalto University School of Science and Technology, Espoo, Finland
e-mail: Petra@PetraBosch.com

D. Šmite et al. (eds.), *Agility Across Time and Space*,
DOI [10.1007/978-3-642-12442-6_15](https://doi.org/10.1007/978-3-642-12442-6_15), © Springer-Verlag Berlin Heidelberg 2010

scale of software systems increases with an order of magnitude about every decade and the architectural, tools, processes and organizational approaches need, to a large extent, be reinvented at the same frequency.

Over the last decade, we can see three main trends drive the increasing complexity of software development [6]. First, the widespread adoption of software product lines [3, 4, 8] causes increasing dependencies between different organizational units that earlier were independently developing their software products and services. Second, the increasing use of global software development teams, where the development of a large software system is spread over two or more continents. This is causing informal or more formal approaches to software process to become significantly less productive due to the inefficiencies of coordination over geographical, cultural and time zone boundaries [see e.g., 7, 11, 12, 16]. Third, there is an increasing popularity of software ecosystems [6, 13], i.e. a company providing a software platform and group of 3rd party developers that provide functionality on top of the platform. The factors complicating software development in this context include the lack of process mechanisms over corporate boundaries and the inherent tension between the interests of the platform company and the 3rd party developers. The focus of this paper is on the second trend, i.e. distributed and global development.

During the 1990s and the early 2000s, the complexity of software development was addressed through large software process efforts such as the Capability Maturity Model [SEI@CMI] that tried to formalize and standardize the development process to increase predictability of resources usage, time and quality. The negative implications of heavyweight process approaches were identified and acted upon by the agile software development community. Over the last decade, several agile software development process approaches have been developed, including XP [1], lean software development [14] and scrum [15, 17]. In the context of smaller scale software development projects, agile development projects have shown significant success. Inspired by the agile approaches, especially for web applications and services, software teams now focus on small team size, short release cycles, ranging from weeks to several times per day, and experimentation in the market place, i.e. the notion of perpetual beta.

Agile development has been widely documented [1, 2] as working well for small (<10 developers) co-located teams. From Agile software literature it becomes clear that agile teams work mainly co-located, have frequent face-to-face contact and highly motivated team members work in self-organized teams. Techniques such as pair-wise programming, daily standup meetings and sprint planning meetings are relying to a large extent on the team being co-located. As a consequence, agile development has shown success especially in small software development projects.

The key topic we address in this paper is the relation between large-scale and agile approaches to software development. All process approaches discussed so far assume what we refer to as an *integration-oriented* approach [6] to software development, i.e. system integration is a major and effort consuming part of the software development cycle as all system components need to be perfectly aligned with each other in order to provide the required system functionality. As a consequence, release cycles, size of software teams, the process overhead, etc. are increasing dramatically over time and grow exponentially with increasing system size. Although

there are application domains where systems need to be highly integrated and the consequences outlined above do not represent a competitive disadvantage, in most domains this is not the case.

The premise that we put forward is that although both traditional software process approaches and agile approaches propose mechanisms to deal with increasing scale of software systems, the fundamental problem is that the coordination cost of taking an integration-oriented, process-centric approach to software development is fundamentally flawed. Process-centric assumes people performing certain tasks as part of the process definition. The inherent assumption is that by formalizing the interactions within and between teams, the pitfalls found in less mature project organizations, e.g. unpredictability, major mismatches between components late in the lifecycle, etc. can be avoided. Experience shows that this is indeed the case, but the price that the organization pays for this is a degree of inefficiency that grows exponentially with increasing system size. The root cause of this inefficiency associated with large-scale software development is the coordination cost between all the teams and individuals involved in the overall software system. Whereas process approaches aim to structure and optimize these interactions and coordination efforts, the consequence is that the symptoms are addressed and not the root cause.

Of the three trends complicating software development that we discussed earlier, we believe that global software development, i.e. distributed development crossing geographic, cultural and time zone boundaries, are particularly affected by the issues discussed so far. This is because coordination efforts, in the end performed by humans, are even more costly in cases characterized by geographic distance, minimal overlap in working hours and cultural differences. Several examples exist where the coordination cost in a global context were a, if not the, major factor in the failure of a major software development effort.

The contribution of this paper is that we propose an alternative: rather than relying on process-centric coordination, we propose the use of the system architecture as a mechanism for coordination and outline how to achieve inter-team coordination. By basing software development on a software architecture that provides decoupling and simplicity, large-scale software development can provide the same efficiencies as small-scale development by providing individual teams, typically associated with a system component, ease of development, independent releasing of components of the system as well as allowing for easy incorporation of external developers and the components developed by them.

The remainder of the paper is organized as follows. In the next section, we discuss large-scale software development as well as a number of definitions. After that we present the case study companies in which we, primarily through participant-observer case study research, studied the challenges of large-scale software development. Subsequently, we discuss the problems of coordination in integration-centric software development approaches. In the next section, we define the architecture-centric approach to coordinating development teams. Finally, we conclude the paper.

15.2 Large-Scale Software Development

Although many development projects are small scale, many if not the majority of software engineers work in the context of large-scale software development. We define large-scale software development along three dimensions, i.e. size, team distribution and specialization. *Size* we define in terms of the number of individuals and teams. The number of individuals ranges from tens at the low end to hundreds or even thousands of engineers. Similarly, the number of teams ranges from a handful to tens or more than a hundred.

The second dimension is *distribution* of teams. We define three levels of distribution, i.e. local, distributed and global. We consider software development local if all teams are located at the same site and could, potentially, meet daily for face-to-face meetings. Distributed teams do not have the ability to frequently meet personally, but can compensate through technological means, e.g. telephone meetings, video conferencing, etc. to have synchronous (same-time, different place) communication. Global teams are located, as the name indicates, around the globe and have very few overlapping working hours during the day. Communication tends to occur primarily through asynchronous means such as email and file sharing. To illustrate the latter, the time difference between California and India is 11.5 or 12.5 hours, depending on the daylight savings schedule. As a consequence, global teams working on the same system have no overlapping regular working hours. Inter-team communication tends to be asynchronous, complemented with individuals at both sides organizing telephone or video meetings during early mornings and late evenings. In Fig. 15.1, we visualize the three types of team distribution.

The third dimension is the degree of *specialization*. In small-scale development, each team member, independent of the job title, is aware of virtually everything

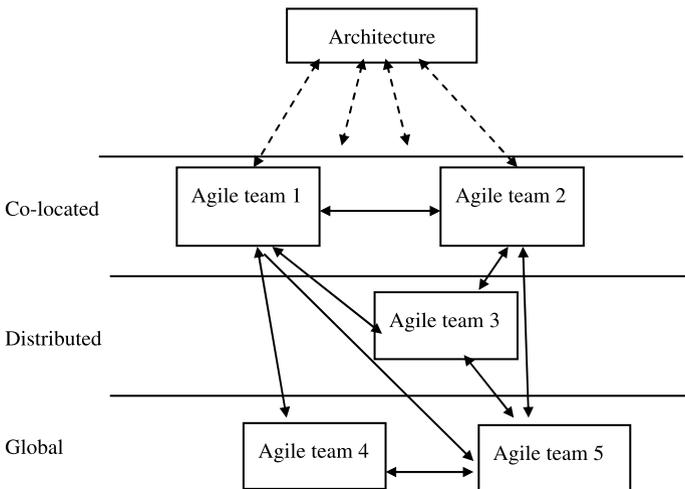


Fig. 15.1 Illustrating local, distributed and global teams

that is going on just by virtue of being part of the team. That, however, does not scale to large-scale software development. Consequently, individuals within the organization need to specialize into specific tasks associated with specific subsystems and information sharing becomes a formal activity with dedicated operating mechanisms associated with it.

Finally, throughout the paper, we use a number of concepts that require a more precise definition. **Coordination** is a consciously organized relation between activities and forces [10], work tasks are divided over actors and the act of is making different people or things work together for a goal or effect. For coordination a number of coordination mechanisms or instruments can be applied like direct supervision, standardizations and interaction or communication. Communication (synchronous, asynchronous and face-to-face) is an important mechanism used for coordination, but other mechanisms for coordination exist, including the use of the software architecture as a coordination mechanism that requires minimal communication between teams. We define **integration** as the *manual* process of combining the components into a working whole. We define **composition** as the *automated* process of combining components into a working system.

15.3 Case Study Companies

The research and approach presented in this paper is based on a participant observation methodology applied by the authors in numerous software-intensive system companies as well as in other industries. The participant observation techniques were applied per case study and individual case study analysis was performed. As a second step, the case study data were compared with help of comparative case study analysis methods [9]. Data was collected in three global organizations by participant observation, interviews and workshops over a period of 3 years per case company (see Table 15.1 for an overview).

15.3.1 Case Company GLOembed

Case company GLOembed is a Fortune 100 company that builds a wide variety of embedded systems for different markets. We mainly focus on the division that develops products for the global consumer market, basically servicing all continents. The business strategy of the company is focused on having a rich set of consumer products in the market, while minimizing the development effort through the application of software product line principles. The size of the software in the products ranges in the several million lines of code. The development teams are distributed across three continents, resulting in global development that requires careful coordination as the company employs a product line approach. Although each product is built from a standard platform, the development of the platform is not centralized, but rather the platform components are owned by distributed teams, but can

still be used, extended and changed by product teams in other locations. The case company does not work with agile teams as such, but has subsystem teams (building components) and product teams (who build products out of subsystems). The teams are primarily co-located, although some are global, and intra-team coordination is mainly performed through same-site and same-place communication and mostly through informal means. Coordination within the team is a relatively simple tasks shared by all team members. Inter-team coordination is performed through architects in whom the lead architect communicates all strategy related aspects to the globally distributed teams.

15.3.2 Case Company GLOtelcom

Company GLOtelcom is a Fortune 100 company developing embedded products, i.e. products that include mechanical, hardware and software parts. The company releases several new products per year and uses a software product line approach to decrease the per product software R&D expenditure. As a consequence a significant part, i.e. more than half, of the software R&D is performed in the central platform organization. The size of the software ranges in the 7 to 15 million lines of code range. The company, being global, has development sites in several locations in Europe, the Americas and Asia, specifically India. The software platform organization is, consequently, also distributed across the world. The organization is transitioning to work more with agile teams (currently 30%). In these agile teams full component responsibility was assigned to a geographically local team in Asia. Development takes place in 2-week cycles; teams consist of 10–20 members and coordinate development efforts mostly through informal means. The head of the team and lead architect coordinate over geographical and architectural boundaries. The team has bi-weekly integration processes with HQ through central architecture teams, integration teams and product management teams. The inter team coordination involves a large amount of communication between many different teams and organization members and units.

15.3.3 Case Company GLOsoftware

Company GLOsoftware is a Fortune 500 company developing software products and services operating, primarily, on personal computers. The company's products address both consumer and business markets and the company releases several products per year, including new releases of existing products and completely new products. The products developed by the company range in the multi- to tens of million lines of code and tend to contain very complex components that implement national and international regulations. Although significant opportunities for sharing between different products exist, the company has organized its development based

on a product-centric approach, i.e. teams are organized around a product and tend to be geographically local. Consequently, little or no sharing takes place between teams. The company works for 50% with agile teams and 50% with TSP/PSP teams, which are fully local (and co-located). It has new product development teams (who have no interdependency with other teams) and component teams in large established products in both Northern America and Asia. The teams are fully co-located in either the US or in Asia and have a local leader. Intra team coordination is performed by 4-week sprints and the normal agile coordination mechanisms such as daily stand-up meetings, product backlog, etc. Coordination between teams is performed centrally by the product management organization.

Table 15.1 Summary of the case studies

Summary of cases	GLOembed	GLOtelcom	GLOsoftware
Number of developers	> 1000	> 1000	> 1000
Domain	Consumer electronics	Telecommunication	Software development
Software development approach	No agile teams. Top down approach – Teams building sets of components – Product teams (set of sub systems of components build into product)	Transition to agile teams (+/-30%). Local teams in Asia with one remote team lead at Head quarters.	50% agile teams and 50% TSP/PSP teams – New product development teams (no inter team coordination) – Component teams in large established products
Size of development teams	20–40 team members	10–20 team members (agile)	5–10 team members (agile)
Location	Primarily co-located development teams Teams all over the world	Main development team co-located with remote team lead. Teams mainly in Europe and Asia	Primarily co-located development teams Teams mainly US and Asia
Coordination within team (intra team)	Teams primarily co-located, but some global. Coordination mainly through informal mechanisms	Co-located teams in Asia, 2-week development and informal coordination. Much contact with lead at HQ in Europe	Teams fully local (co-located) in either US or Asia, with local leader. Sprints of 4 weeks periods, daily stand-up meetings, product back-log, etc.
Coordination between teams (inter team)	Coordination and communication through architects. Lead architect communicated to all teams on strategy related aspects	Bi-weekly integration process. – Central architecture team – Integration team – Product management teams Many people involved.	Central coordination between teams by product management organization.

15.4 Coordination and Integration Inter-team Challenges

From our cases we found that the smaller (local) teams were able to coordinate their work rather efficiently and effectively as is confirmed by agile software development literature. However, the main problems we found in the case studies were challenges between inter-team communication and inter-team coordination especially for large-scale software development. These challenges can be placed on a continuum on which on one side local inter-team coordination is placed and on the other side of the continuum the global inter-team coordination is situated. The inter-team coordination challenges increase when teams have to coordinate over different time zones, cultures and countries (global).

Below we discuss the main problems we found from the case studies.

1. Top-down approach challenges or process-centric approach problems related to inter-team interaction.
2. Interaction problems.

15.4.1 Top-Down Approach Challenges

Process-Centric Coordination All three cases applied a process-centric approach for inter-team coordination for all phases of the software development lifecycle, including road mapping, requirements, dependency management during development, API evolution, integration and release management. Case study GLOembed applied a model in which only architects between the teams communicated with each other and a lead architect traveled to all the different team sites to communicate about the strategic plans and road maps. Case study GLOtelcom had local teams in India, but the lead architects were at headquarters. Furthermore, road mapping, product management and integration were done by numerous meetings that either took place in person at the headquarters, requiring all remote team representatives to travel, or through teleconferencing, requiring remote team members to attend outside work hours. Case GLOsoftware had a central organized inter-team coordination process lead by a central product management department who communicated to all the different component teams. All these teams were dependent on a central and top-down unit for inter-team coordination, which implied challenges in amount of communication (case GLOtelcom and GLOsoftware) and coordination needed for integration, and high dependency on one lead architect (GLOembed). In all cases, the amount of effort that was spent on non-value adding activities was very high and increasing over time as more and more items were identified that required collaboration between teams.

Integration Costs All three cases applied some sort of process-centric approach for coordinating work between teams for large-scale software development. However, we found that all cases had high and unpredictable product integration cost.

We observed in all case study companies that during product integration, incompatibilities between components are detected during system tests and quality attributes break down in end-to-end test scenarios. This causes a costly and unpredictable integration process that, being at the end of the development cycle, causes major difficulties at the affected companies.

Coordination and Communication Costs Between Teams A problem observed in all case study companies is that when decoupling between shared software assets is insufficiently achieved, excessive coordination cost between teams are one outcome. One might expect that alignment is needed at the road mapping level and to a certain extent at the planning level. When teams need to closely cooperate during iteration planning and have a need to exchange intermediate developer releases between teams during iterations in order to guarantee interoperability, the coordination cost of shared asset teams is starting to significantly affect efficiency. Case study GLOtelcom showed an example where communication and coordination costs were very high due to a large amount of integration meetings between all the different involved units for large-scale software development.

Unintended Resource Allocation Resource allocation is a tool used by companies to align resources with the business strategy. In practice, however, at two of the case study companies, i.e. GLOtelcom and GLOsoftware, teams frequently assign part of their resources to other software components and their associated teams. The reason is that they are dependent on the other components to be able to get their own functionality developed and released. One can view this as a lack of road mapping activities and inter-team coordination. The consequence is again, that the coordination costs between teams easily become excessive, resulting in a general perception in the organization that significant inefficiencies exist.

Insufficient Pre-iteration Cycle Work In some of the teams in case company GLOsoftware, features that cross component boundaries were underspecified before the development cycle started and were “worked out” during the development. In practice, this requires close interaction between the involved teams and causes significant overhead that could easily be avoided by more upfront design and interface specification. A consequence of this approach is that it builds an “addiction” between teams in that there is a need for frequent (daily) developer-to-developer drops of code that is under development in order to avoid integration problems later on. This, in turn, often results in largely manual testing of new functionality because requirements solidify during the development cycle and automated tests could not be developed in time.

15.4.2 Interaction Problems

Global Interaction Problems Between Teams Interaction between global teams implies more challenges due to time zone differences, cultural and language differences and, often, different work practices. For example, in one organization that we

worked with, case company GLOtelcom, teams were geographically split, with the team lead architect and senior engineers located at the main site of the organization in Europe and the remaining engineers in a remote site in India. This required significant communication taking place over geographical boundaries resulting in very inefficient development processes as well as a de-motivated team at the remote site, due to a lack of autonomy and responsibility of the remote site. Another example is case company GLOsoftware in which teams from the US cooperate with teams from Asia with a 12.5 hour time difference. Inter-team communication and coordination can only happen asynchronously or by traveling to the different locations to meet face-to-face. In GLOembed the lead architect travelled to all the global sites to visit the teams in person to discuss road mapping and strategic decisions.

Maintaining Motivation in Remote Teams In all case companies, we observed behavior at the main site of the organization that would keep the most interesting and strategic work at the main site and outsource the routine and less strategic work. In addition, there was a strong desire to maintain control over work that took place at the remote sites and to exercise that control through direct supervision of remote individuals and teams. This was caused both by a sense of protectionism at the main site, where work at the remote site was considered threatening. It also was a consequence of applying the same operating mechanisms that are applied locally, where frequent face to face contact is not experienced as supervision, in a global context where the interaction tends to become much more formal. The consequence was significantly reduced motivation and retention in the remote sites. This may turn into a self reinforcing system if work performed at the remote sites is of insufficient quality, or at least perceived to be, which further reduced trust in the main site to delegate work to the remote site.

Low Productivity In case study company GLOembed and GLOtelcom, the productivity of teams as well as of the overall system integration was very low in the cases where teams were internally distributed and where the coordination between teams was very process-centric with extensive coordination taking place during every phase of the lifecycle. Especially during systems integration, where the software assets from the various teams are brought together, many incoherencies were identified, despite the coordination efforts during the development process.

Table 15.2 presents a summary of the observed problems on inter-team coordination of both local teams compared to organizations with global agile teams that need integration between the teams.

15.5 Coordination Through Architecture

Throughout the chapter, we have presented the viewpoint that the root cause of the inefficiency associated with large-scale software development is concerned with the amount of coordination that is required between teams. The problems discussed earlier in the chapter are either a direct consequence of that root cause or can be traced

Table 15.2 Observed problems with process-centric coordination approaches between agile teams

Observed problems in inter-team coordination	Local		Global
Process-centric coordination	Relatively inexpensive due to largely informal, face-to-face communication. Broad interfaces between teams		High costs – Dependency on architects/central units for inter-team coordination tasks
Integration cost	Lower to medium cost. Productivity and outcome higher (faster)		High cost Low productivity
Communication & coordination cost	Lower communication and coordination costs. – Daily face-to-face or synchronous mediated interaction		High communication and coordination costs. Very costly – Inconvenience due to time differences – Quality of interaction lower – Technology solutions
Interaction problems	Interaction between teams easier because of close proximity, same time zone and similar language, culture and work practices		Problems with time zones, cultural and language differences, differences in work practice. Influence coordination and communication cost

back to it. Addressing this root cause is conceptually very simple: remove all need for inter-team coordination. That would allow small, agile teams to develop and re-release independently and increase efficiency of software development tremendously. However, the teams are still building solutions that are part of a larger system and therefore cannot be completely independent. The approach that we, based on our experience with Web 2.0 companies and software ecosystems, describe here is to move any remaining coordination needs from the process level to the architecture. This, in effect, replaces manual work with an automated solution.

The cost associated with process-centric coordination is much higher in a global context than in a local context due to the communication inefficiencies. Development approaches that rely on significant inter-team communication perform poorly in global and distributed contexts. The amount of coordination between teams can be reduced to a quite significant extent compared to what traditional software development approaches dictate. Below, we discuss the coordination needs for each stage of a traditional software development lifecycle.

15.5.1 Road Mapping

Traditionally, the road mapping process outlines high-level features and assigns these to releases of a large system. Assuming a release frequency of 6 to 12 months,

every release contains several new features. The road mapping process requires the organization to decide on the relative priority of the things that it could build. In order to decide on this, the effort associated with each high-level feature needs to be estimated. The effort estimations are naturally rather coarse and lack accuracy, which often affects the latter stages quite significantly.

The importance of an accurate ROI (return on investment) and effort estimation for each high level feature causes most organizations to involve people from virtually every function and team involved in the development, sales and deployment of the system. Especially for large systems, this often means that several tens of people are involved.

In the architecture-centric approach the organization translates its business strategy into a number of domains of functionality where it wants to see significant improvement. The teams take these domains as input for determining what to build in the next iteration. However, as discussed in the next section, the organization does not plan and order the exact functionality to be built but instead relies on the teams to optimize.

For the organization, it means giving up control and predictability in terms of the functionality delivered. However, it is important to realize that the notion of control and predictability tends to be an illusion in most companies.

15.5.2 Requirements

In traditional development, at the start of every iteration the high-level features assigned to this iteration are translated to more detailed system level requirements. These requirements are, in turn, translated to component level requirements. At this stage the overlapping with other activities starts in earnest as the process of translating system level requirements to component level requirements requires active involvement of the architects and team leads to make sure that the requirements allocation is appropriate and that the effort estimations are supported by the teams.

In the architecture-centric approach, there is no centralized requirement management process. Each team, which is associated with a component in the system, evaluates the domains in which progress is desired, complements that with its own customer understanding and announces to the organization what it intends to release at the end of the iteration. There is no coordination of requirements and there is a risk that more than one component team attacks related or similar functionality. On the other hand, because there is no coordination between teams, no effort was lost on non-value adding activities.

15.5.3 Architecture

The next activity in development is to determine the impact of the new requirements on the architecture and to design the changes to the architecture. This typically results in added and removed components, but the primary area of concern is often the impact on interfaces between existing components and, by extension, the teams responsible for these components.

As we discussed earlier in the chapter, in traditional software development, the architecture is often underspecified and teams are at liberty to develop interfaces between their components during development in mutual discussion. This may seem efficient as it allows for working in a decentralized fashion, our research at the case study companies as well as with other companies shows that architecture is the one area where discipline needs to be enforced. For every problem not handled by the architecture, a process coordination mechanism needs to be put in place to allow teams to release the system.

In architecture-centric development, component teams not only announce the requirements but also the changes to their component from an external perspective, including interfaces to be added, deprecated and removed by the end of iteration. A separate team manages the architecture, with a focus on compositionality and backward compatibility.

15.5.4 Development

The fourth activity is development. The case study companies had, to a significant extent, adopted agile development methods with four to six week development cycles. Ideally, development takes place in isolation from other teams so that each team can be as effective as possible. In practice, the teams need to spend a lot of time aligning their development effort with other development teams, test teams and the integration team.

The high coordination cost was caused by several of the issues discussed earlier in the chapter, but two of the key drivers were the lack of architectural specification and concurrent development of functionality. Teams spent too little time during the preparation of the iteration on analyzing and designing detailed changes to the component interfaces with the intention to “work it out” during the development cycle. Especially in global development this is particularly inefficient. The second main cause of coordination overhead is concurrent development. System-level features often require changes in multiple components and these changes typically have dependencies on each other. Concurrent development requires teams to interact during the development stage to work out compatibility issues and detailed assignment of responsibilities.

Architecture-centric development is concerned with facilitating independent development by component teams and to minimize the number of unproductive hours spent on coordination while maximizing the amount of productive hours. As the team has announced the interface changes, knows what backward compatibility is required, knows what functionality it wants to build and the other component interfaces to develop against, this stage should allow the team to focus solely on development. One of the principles that need to be enforced in this context is that no team can initiate development on functionality that is dependent on functionality that is under development by another team. Although this at first may seem to slow development as the implementation of a system level feature requires multiple iterations depending on the number of dependencies, in practice the removal of coordination cost and the short cycles for most agile teams outweighs any benefits that may be achieved by concurrent development.

15.5.5 Integration or Composition

In traditional development, the development of the next version of the components is followed by an integration phase. Here the fruits of the work of the various development teams are brought together and integrated in a product or platform release. As discussed in the problems statement, in the case study companies, the integration stage is very effort consuming and unpredictable. All case study companies used forms of continuous or frequent integration. However, the SCM (source control management) and test infrastructure did not allow for full coverage and hence the companies still used an explicit integration and validation phase before releasing the new product system to market.

The integration phase is especially painful in global software development as there is enormous need for interaction between the integration team and all of the component teams. During system testing, many issues are found that require collaborative resolution between teams. Although the amount of interaction needed may be limited, in global contexts there often are significant delays due to time zone differences, causing many issues that could be resolved in minutes or a few hours to become part of a daily rhythm instead.

In architecture-centric development, there is no integration phase, but instead the system is focused on composition. Each component team releases frequently, but uncoordinated with other teams. When a component team releases, its component has to pass the automated SCM and test system. The automated test system is improved in response to any problem that manages

to get through the system and is only surfaced after deployment. As a consequence, over time the quality of the validation reaches a very high level. The traditional approach is to put process steps in place to avoid problems to occur, but this requires coordination and manual effort. This additional focus on the automated SCM, test system and deployment infrastructure removes the need for an unpredictable and effort consuming integration phase and allows teams to release their components independently.

15.5.6 Architecture-Centric Software Engineering

Architecture-centric software engineering focuses on minimizing the inefficiencies associated with traditional process-centric development. The approach adopts a set of principles that is different and often initially uncomfortable in corporate contexts. However, there is of course a clear parallel to the development approaches found in the open-source software communities.

The key enabler for architecture-centric software engineering is to minimize dependencies between components. Although this central to architecture design, architects often de-prioritize decoupling to achieve other attributes. In [5], we present the notion of software ecosystems where architecture decoupling is paramount for its success. The principles it introduces are valuable in this context as well.

The concerns in a corporate context are often related to the loss of control over R&D investment, resource allocation and product roadmaps. Our experience from the case study companies as well as other organizations is that the perception of control often is an illusion. Either the R&D organization operates at such a low expectation level that any organization can meet it, or plans and milestones are frequently missed in unpredictable ways.

Architecture-centric software engineering removes so many inefficiencies from the software development process that the output of the organization is much higher, even if senior management has less visibility into the operational issues in the R&D organization.

Although none of the case study companies has implemented all aspects of the architecture-centric software engineering approach, each employs some of the practices. The consequences of globalizing their software development while interested in adopting more agile development approaches necessitated each of the case study companies to change some of their, initially process and integration-oriented, practices and adopt a more architecture-centric approach. Based on our research at these companies, Web 2.0 companies and in the context of software ecosystems, we are convinced that the presented approach provides enormous benefit to organizations that adopt it.

Practical Tip: Illustrate the lack of predictability in large-scale software development by collecting and analyzing historical data. In most companies, there is a significant gap between plan and outcome. This data can then be used to break the illusion of control and to create an opening for experimenting with a new approach. Once the experiment is approved, make sure to deliver real business value as soon as humanly possible and collect data on relevant metrics, e.g. productivity or time to customer of new functionality. Select comparable development efforts using the traditional approach to support the transition from a belief that the new approach is better to a quantitatively supported position that the new approach is superior.

15.6 Conclusions

Over the last four decades, software engineering has continuously evolved to address the continuous and enormous increase in complexity due to sheer system size, the complexity of the application domains and level of interaction required with other embedded and IT systems. The case study companies reported on in this paper have been very successful in applying traditional software development approaches to their product development and have, as a consequence, seen significant growth.

With increased globalization of software development and the increasing popularity of agile software development approaches, it has become blindingly obvious that a process-centric approach to large-scale software development over time results in unmanageable complexity and unacceptable inefficiency. In the paper, we discuss several problems, categorized in four categories, i.e. process-centric coordination, integration cost, communication and coordination cost and interaction problems. These problems can largely be attributed to one root cause: dependencies between components in the architecture and the teams responsible for these dependencies.

Based on our research with the case study companies, but also with a several other companies as well as software ecosystems, we propose an alternative: rather than relying on process-centric coordination, we propose the use of the system architecture as a mechanism for coordination and outline how to achieve inter-team coordination. By basing software development on a software architecture that provides decoupling and simplicity, large-scale software development can provide the same efficiencies as small-scale development by providing individual teams, typically associated with a system component, ease of development, independent releasing of components of the system as well as allowing for easy incorporation of external developers and the components developed by them.

The contribution of the paper is twofold. First, it presents the results of a case study into the implications of applying process-centric, integration- approaches in large-scale oriented software development based on longitudinal case studies at

three large organizations. Second, it presents architecture-centric software engineering as a novel approach that combines the best practices from these companies, as well as from companies in the Web 2.0 and software ecosystem industries.

References

1. Beck, K. (1999). *Extreme programming explained: Embrace change*. Boston: Addison-Wesley.
2. Boehm, B., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Boston: Addison-Wesley.
3. Bosch, J. (2000). *Design and use of software architectures: Adopting and evolving a product line approach*. London: Pearson Education (Addison-Wesley & ACM Press).
4. Bosch, J. (2002). Maturity and evolution in software product lines: Approaches, artifacts and organization. In *Proceedings of the 2nd software product line conference (SPLC)* (pp. 257–271), San Diego, USA, 19–22 August 2002.
5. Bosch, J. (2009). From software product lines to software ecosystems. In: *Proceedings of the 13th international software product line conference (SPLC 2009)*, August 2009.
6. Bosch, J., & Bosch-Sijtsema, P. M. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, 83, 67–76.
7. Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*, 1(2), 22–29.
8. Clements, P., & Northrop, L. (2001). *Software product lines: Practices and patterns*. Boston: Addison-Wesley.
9. Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of Management Review*, 14(4), 532–550.
10. Hatchuel, A. (2001). Coordination and control. In A. Sorge & M. Warner (Eds.), *The IEBM handbook of organizational behavior* (pp. 320–339). London: Thompson Business Press.
11. Herbsleb, J. D., & Moitra, D. (2001). Global software development. *IEEE Software*, 18(2), 16–20.
12. Kraut, R., Steinfield, C., Chan, A. P., Butler, B., & Hoag, A. (1999). Coordination and virtualization: The role of electronic networks and personal relationships. *Organisation Scientifique*, 19(6), 722–740.
13. Messerschmitt, D. G., & Szyperski, C. (2003). *Software ecosystem: Understanding an indispensable technology and industry*. Cambridge: MIT Press.
14. Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Boston: Addison-Wesley.
15. Rising, L., & Janoff, N. S. (2000). The Scrum software development process for small teams. *IEEE Software*, 17(4), 26–32.
16. Sanwan, R., Bass, M., Mullick, N., Paulish, D. J., & Kazmeier, J. (2006). *Global software development handbook*. New York: CRC Press.
17. Schwaber, K. (2001). *Agile software development with Scrum*. New York: Prentice Hall.

Chapter 16

Considering Subcontractors in Distributed Scrum Teams

**Jakub Rudzki, Imed Hammouda,
Tuomas Mikkola, Karri Mustonen,
and Tarja Systä**

Abstract In this chapter we present our experiences with working with subcontractors in distributed Scrum teams. The context of our experiences is a medium size software service provider company. We present the way the subcontractors are selected and how Scrum practices can be used in real-life projects. We discuss team arrangements and tools used in distributed development teams highlighting aspects that are important when working with subcontractors. We also present an illustrative example where different phases of a project working with subcontractors are described. The example also provides practical tips on work in such projects. Finally, we present a summary of our data that was collected from Scrum and non-Scrum projects implemented over a few years. This chapter should provide a practical point of view on working with subcontractors in Scrum teams for those who are considering such cooperation.

16.1 Introduction

In this chapter we discuss industrial experiences in organising distributed Scrum teams that include members from subcontracting organisations. We present specific

J. Rudzki (✉) · T. Mikkola · K. Mustonen
Solita Oy, Satakunnankatu 18 A, 33210 Tampere, Finland
e-mail: jakub.rudzki@solita.fi

T. Mikkola
e-mail: tuomas.mikkola@solita.fi

K. Mustonen
e-mail: karri.mustonen@solita.fi

I. Hammouda · T. Systä
Tampere University of Technology, Korkeakoulunkatu 1, 33101 Tampere, Finland

I. Hammouda
e-mail: imed.hammouda@tut.fi

T. Systä
e-mail: tarja.systa@tut.fi

context of working with subcontractors in such teams. We discuss details of our experiences in subcontracting [1] and in Scrum projects [2] that have been reported in our previous publications. However, in this chapter we focus on the practical aspects of cooperating with subcontractors in distributed Scrum teams.

The practices that we discuss should be particularly helpful for companies that are planning to use subcontractors in their distributed Scrum teams. However, those who already have such teams, can use all or selected recommendations in the later parts of this chapter. First we present, in Sect. 16.2, the process of selecting subcontracting partners we used. Next in Sect. 16.3, we discuss agile practices and tools that have been used in our distributed and subcontracted project teams. Those practices and tools are rather generic agile practices, but their specific aspects, for example quick feedback, are particularly important in the context of distributed teams with subcontractors. In Sect. 16.4 we go through a life-cycle of an example distributed project phases where we present how the agile practices and tools are used when working with subcontractors. Finally, in Sect. 16.5, we conclude this chapter with a summary of findings and possible future work. However, before going into details we should present the company (Sect. 16.1.1), which is the context for this work, and present our methodology (Sect. 16.1.2) and main results (Sect. 16.1.3) as an executive summary for the readers who are not interested in the actual details.

16.1.1 Company Context

As the experiences that we discuss have been gained in a context of a specific company, we will first present this context. Our experiences have been obtained through a few years of cooperation with subcontractors at Solita.¹ Solita is a Finnish software service provider (SSP) that specialises in providing high quality software services in various domains. Since 1996 Solita has offered its services to customers in different domains, ranging from media, telecommunication, to public sector institutions and others. The diversity of customers and provided solutions demand very close cooperation with end customers, which requires a special approach to team organisation and choice of processes used. Additionally, the company has been changing internally over the years and has grown into a medium size company of 150+ specialists with two offices in two cities in Finland. These details give some perspective to the experiences we report on.

16.1.2 Methodology

Our findings on subcontractors in Scrum teams are based on our experience and the study we have conducted to find out the differences between Scrum and non-Scrum

¹www.solita.fi.

projects. The summary of that study has been already published [2]. To obtain the data we prepared five questions following the Goal Question Metric (GQM) [3] methodology. We found this method to be most suitable for finding out how Scrum teams performed comparing to other teams. In this chapter we focus only on the study parts that are most relevant for the subcontractors involvement in Scrum projects.

Our GQM questions were as follows:

- *Q1: Do the current agile practices benefit projects?* We used three metrics Customer Satisfaction, Profitability, and Team Performance to answer this question.
- *Q2: Does the customer's direct involvement in the project benefit project success?* We distinguished three levels of customer involvement (none, partial and full involvement) and grouped project results based on this metric.
- *Q3: Is the communication different in agile projects?* We used two metrics the Communication Factor (time spend by the whole team for communication tasks) and Project Manager time (time spent for management tasks).
- *Q4: How to adapt agile practices to suit commercial needs?* This question was answered based on interviews with project leaders of all the investigated projects.
- *Q5: What kinds of projects suit agile practices?* This question was based on the project results and interviews with project leaders.

The data was collected based on interviews with Solita's project leaders and data was also obtained from IT systems that track project work time and financial data. We would like to clearly indicate that the data we collected was gathered with as much care as it was possible at the time, but it is not a perfect collection. Our study is limited to only one specific company. Therefore, our study is placed in the context of a medium size software service provider. Cases dealing with larger distributed teams can be found in literature, for example in a report on distributed teams developing Windows Vista [4].

Additionally, the collected data has also limitations regarding its quality, which might have affected metrics used for measuring customer satisfaction and team performance in particular. Both of the metrics were based on the subjective judgement of the project managers. Despite those limitations the data used as indicative reference point may be useful for other researches or practitioners who seek some references from similar contexts.

16.1.3 Main Results

All of the analysed projects amounted to 18 projects realised between 2006 and 2008. From those projects 8 projects were Scrum projects, which are most relevant for this study. We present more detailed data in the [Appendix](#), while in this section results mostly relevant to subcontractors in Scrum teams are briefly discussed.

Findings to the question *Q1: Do the current agile practices benefit projects?* showed that Scrum projects performed in general better than traditional projects.

The number of subcontractors did not seem to be a factors impacting projects success in the analysed cases.

The next question that was relevant to subcontractors was *Q3: Is the communication different in agile projects?* An answer to this question showed that both Communication Factor and PM time were usually higher in projects with one subcontractor in a team. That can be explained by a need of additional formal communication with a person who is on a remote side. In general we found that communication factor and PM time increased in Scrum projects, but the increase was not very high (within a few percent points) comparing to traditional projects.

The question *Q4: How to adapt agile practices to suit commercial needs?* is the most relevant to work with subcontractors. The answers included observations and suggestions about the practises used in projects. Those suggestions and observations are presented in details in Sect. 16.3 and illustrated in the example presented in Sect. 16.4.

Finally, in the question *Q5: What kinds of projects suit agile practices?*, which was based on data analysis and interviews, we did not find any absolute limits for usage of Scrum and subcontractors in various project types. Scrum projects performed well even in cases where the teams included subcontractors and were distributed across more than two locations. Most of the Scrum projects were implementation projects where Scrum and subcontractors were used during development phase.

Generally, our findings presented in this chapter indicate better performance in Scrum-driven projects comparing to more traditional methodologies. In the context of work with subcontractors the quick feedback and organised communication seem to be the success factors that help project performance. We also discuss a subcontractor selection process where among other aspects the compatibility in terms of methodologies used and culture are criteria taken into account during the selection. Finally, we provide examples of tools that can be used in distributed projects to provide easy access and communication channels between team members from different organisations.

16.2 Subcontractors in an SSP Company

Having defined the company profile in which we operated we can discuss the processes we used for selecting subcontractors. The selection processes are important from the cooperation with subcontractors point of view that we describe as an example project walkthrough in Sect. 16.4.

The selection of a suitable subcontracting partner is especially important in the case of close cooperation between the development team that includes (or consists of) subcontractors and the end customer. In Solita we used a subcontractor selection process that has been built to suit our specific needs. The process was based on our experiences and existing best practices, which for example include a study on the quality in virtual teams and culture aspects of such teams [5], which was based on sourcing model eSCM-SP [6, 7]. We have already reported details of this process [1], and therefore in this section we focus on the elements that are most relevant to the creation of distributed teams with subcontractors.

16.2.1 Why Subcontractors?

There are many reasons for using subcontractors, the main ones include access to specialists, flexibility in team creation without the need to temporarily extend own staff, and potential differences in costs. Usage of specialists from other software organisations, namely subcontractors, requires careful selection of those partners as well as good team organisation. Subcontracting in a software service provider (SSP) company differs from that of product-orientated companies. The main difference is the direct interaction of subcontractors with the end customer, which usually does not take place in the case of product development.

In the case of SSP, the company serves its customers by delivering customised solutions specific to customer's needs. These solutions may differ in technology and nature of the solution, which may be a software implementation, but it can also be specialised consulting service provided to the customer in the area of the expertise of the SSP. However, in any case a team developing a solution for a customer works very closely with the customer. In that setting the team members, including the subcontractors, must be suitable for such a work environment. This is one of the reasons why Solita carefully selects subcontracting partners.

16.2.2 Distributed Development Stakeholders

Teams that work in the context of SSP companies generally involve a few stakeholders whose roles we should discuss in order to understand their roles in the distributed development teams. We can list three main stakeholders involved directly in development of specific software solutions:

- *Software Service Provider* is the party that orchestrates the whole development process of a solution. SSP is responsible for contacts with other parties (i.e. customers and possible subcontractors). SSP is responsible for delivering a solution that the *Customer* expects. SSP plays the central role in the case of customised solution development.
- *Subcontractor* is the other specialised software organisation who provides their experts to SSP's teams. The subcontractor takes the responsibility for their staff but does not have to be directly involved in specific solution development as a whole. Only selected experts from *Subcontractor*'s team are directly involved in development.
- *Customer* is the ordering party who expects a specific solution to help its business. The *Customer* may be involved in the solution development to a different degree, depending on the needs and expectations. In the case of teams consisting of subcontractors the customer is usually not interested in the details of the team organisation, but the customer is very interested in the results of the team work.

These three stakeholders interact during the solution development process. Frequently, all of these stakeholders are distributed, which contributes to the complexity

of cooperation. Furthermore, in some cases the stakeholders' roles may be mixed, for example, the original SSP organisation may be hired by another SSP. In that case the original SSP is a subcontractor to the hiring SSP, while the original SSP still has its own subcontractors. This may make the organisational complexity difficult to grasp, but the basic roles can still be recognised and the processes and tools specific to given role are applicable.

16.2.3 Subcontractor Selection Process

The subcontractor selection process used at Solita consists of a few stages. The process' steps are depicted in Fig. 16.1. First, we performed an *initial search* of all possible candidate companies. Already at this stage the search was limited by location and technology. In our case the location was limited to a few European countries that from the Finnish perspective can be regarded as nearshore locations. This location limitation was imposed in order to have a possibility of arranging face-to-face meetings quickly, and additionally to reduce timezone problems in the case of every day communication.

The next step was *sending initial offer requests* to companies that were selected from the results of the initial search. Then the replies were analysed for basic company details and possible pricing provided by the potential subcontracting partner. Then *arranging interview* could start. As preparation for the interview, the potential subcontractor companies were sent a questionnaire with questions about the candidate's organisation. If such a company has experience with agile techniques, or at least some of them, it is likely that cooperation with the company at that level will be more straightforward than in the case of companies unfamiliar with such techniques. Additionally, our questions were related to typical projects done at the candidate company. We asked, for example, about project size, typical length of projects, roles in the projects and customer types. Answers to those questions provided information about the candidate company and its suitability to work in SSP teams. We conducted the actual interviews with candidate companies as teleconferences. We also discussed in more details the answers to our questionnaire at that stage giving us clarifications to answers that might not be clear after reading the questionnaire answers.

After conducting the interviews we selected a few companies that seemed to be the most suitable candidates and we arranged an on-site *meeting in the candidate's premises*. One of the reasons for meeting in the candidate office was to see the environment in which the people work there. For example, the Internet connection quality, open or closed office space, and equipment can provide some indication of company infrastructure and the culture, which both are relevant in the case of cooperation with SSP's projects. The visits also gave us a chance to talk directly to specialists from potential subcontracting partners.

Finally, after conducting the interviews and analysing the data the subcontracting partners could be chosen. The partners selected over a certain period of time are

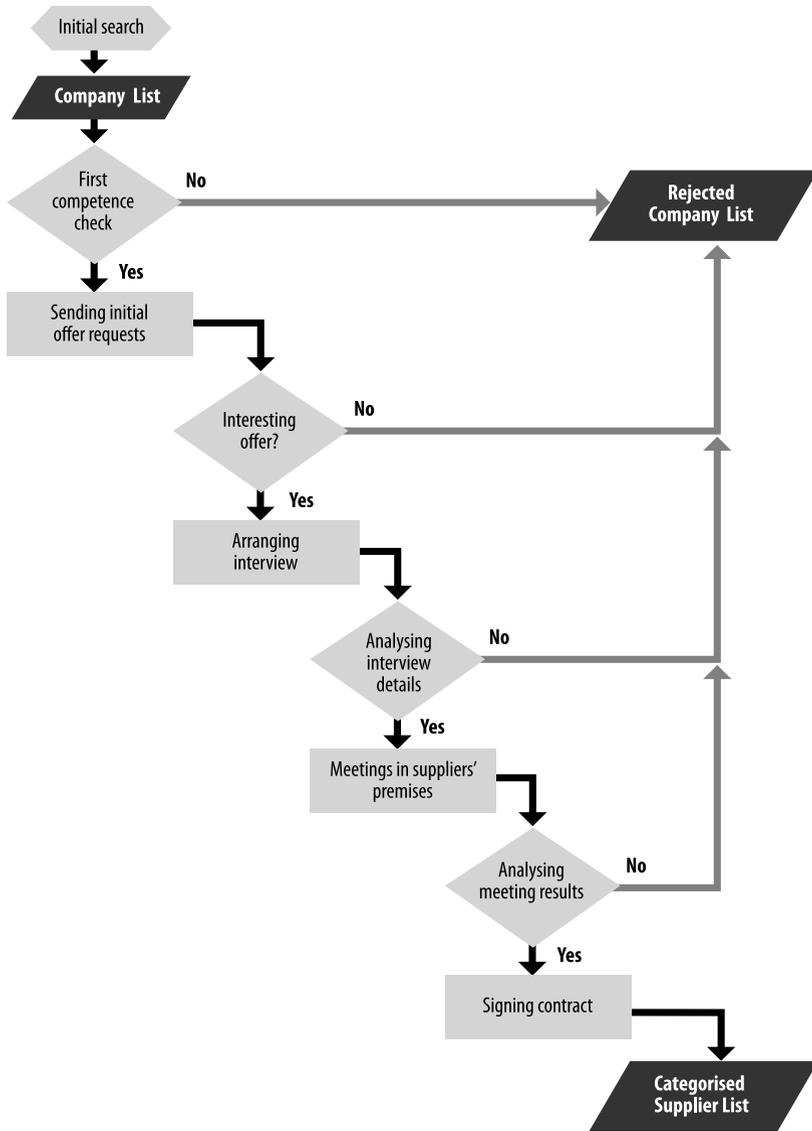


Fig. 16.1 Subcontractor selection phase [1, p. 227]

likely to differ in terms of their technological expertise, but they are likely to have similar cultures. Further cooperation will reveal possible differences and areas for improvements. We present an example of cooperation with subcontracting partners in a project in Sect. 16.4.

16.3 Subcontractors in Scrum Teams

Before going into details of software development in distributed subcontracted Scrum teams, we discuss different agile practices and tools used in projects. We focus on Scrum methodology, but we also discuss other agile supporting practices used in Scrum projects. These practices and tools can be used together or selectively depending on needs. However, certain practices should always be followed in the case of agile projects. There are many practices and tools that could be discussed, however, we are focusing on the ones that have been used at Solita. The practices and tools we discuss have been gathered based on the literature reports, experiences of our customers, and our own observations over years [2]. A good overview of agile practices in a distributed team provides, for example, an article by Martin Fowler [8].

16.3.1 Scrum

The most prominent agile practice used by us is *Scrum* methodology [9]. Scrum has been researched rather extensively in recent years, including the initial methodology description by Schwaber [9] as well as later applications in distributed teams reported, for example, by Sutherland et al. [10] or by Paasivaara et al. [11]. This methodology consists of a few elements that practically help to organise projects. One of the most important aspects of Scrum is a quick feedback loop, which benefits projects in cases where corrective actions must be taken. A team performs work according to a feature list defined in a *product backlog*. The features reflect certain functionalities that the end customer wants to have in the final product. Please note that in this case a product does not have to refer to a real marketable product, but rather a tailored software solution, which helps the customer to achieve certain business goals.

The features are implemented in an order determined by the priorities set by a *product owner*, who is responsible for setting the priorities according to the business value of particular features. The *product owner* selects features that should be implemented in a coming *sprint*. A *sprint* is a period of time dedicated to development of selected features.

An overview of Scrum activities in a project is presented in Fig. 16.2. For each *sprint* there is always a planning session when the implementation scope for the *sprint* is decided. Then status meetings are held on daily basis. At the end of a *sprint* the team demonstrates the implemented features. Also then the past *sprint* is analysed as a retrospective and all team members have an opportunity to comment on the work done. Retrospectives are important in distributed teams as they encourage open communication. Finally, either the development continues in following sprints, or it is finalised with a delivery.

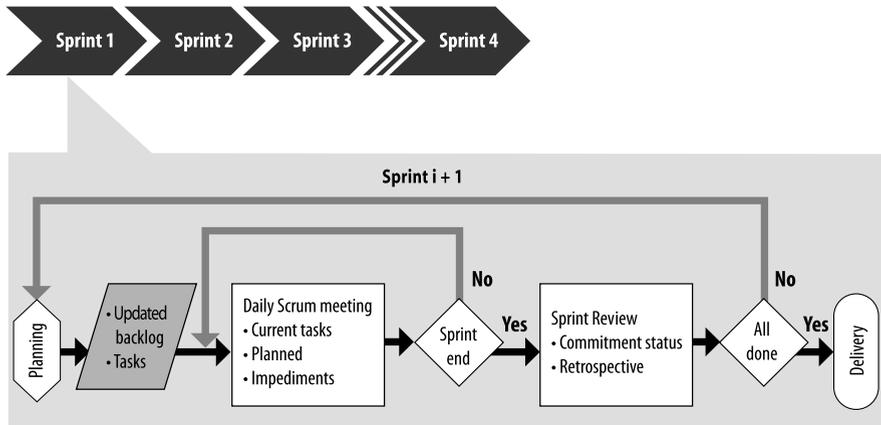


Fig. 16.2 Scrum activities overview

Practical Tip: Our experiences show the usefulness of Scrum in distributed teams with subcontractors. Scrum practices are especially useful in such a context as they provide important feedback loops. One is a daily feedback provided in the whole team on the progress of the development, so that the state of a project should be well known to the whole team regardless of their location. The other feedback is provided at the end of the sprint when the sprint has been analysed. This quick feedback gives a Scrum team the opportunity to take corrective actions before the project goes too far off the planned track. Finally, Scrum renders subcontractors as equally involved in the team activities as the team members from the SSP organisation.

16.3.2 Communication

Communication is extremely important especially in the case of distributed teams, which cannot benefit from direct informal communication as occurs in the case of co-located teams. Scrum defines a few meeting types (i.e. daily meetings, planning and demonstration sessions). Naturally, in addition to these meetings other meetings are often arranged, e.g., special workshop sessions on specific topics, discussions with customers, etc. In all these cases, the practical arrangements of the meetings can be done in different ways.

Face-to-face meetings are most optimal for the efficiency and quality of communication, but such meetings frequently held may not be feasible in the case of distributed teams or customers located in remote locations. In these cases telecommunication tools can be used as a replacement for face-to-face meetings. Of course, a phone teleconference is a natural choice, but it can be replaced by Voice-Over-IP

solutions, which are typically more cost efficient. In addition to voice communication, videoconferencing tools can be used. There are many such solutions, which often offer very good quality. Videoconferencing can be a really good replacement for face-to-face meetings, but its use is limited by the infrastructure of the participating parties. Therefore, we find videoconferencing to be working best between selected locations (e.g., company offices), where the necessary equipment has already been installed. In addition to high-end hardware-based videoconferencing solutions, different software clients can be used. Such clients usually also support other functions, in addition to audio and video. A very useful feature during teleconferences is a shared screen where presentation, or other material can be shared.

Formal meetings that are the official channel of communication, even in the case of frequent Scrum meetings, may not be sufficient. That is why we use additional tools that at least try to enable less formal communication within distributed teams. Instant messaging tools, which often are the same tools as the ones used for voice or video communication, can be a good substitute of less formal communication. Usage of IM in projects is not a new concept and usage of IM in distributed teams has been reported in literature, for example by Herbsleb et al. [12]. In Solita we use Skype² as the primarily teleconferencing and instant messaging tool and LifeSize³ for videoconferencing.

Practical Tip: A development team can use a common chat where all the team members can easily exchange options, notify about changes, ask questions, and see availability of other team members. A chat can be used more willingly by subcontractors from other locations than phone, as it is less disruptive and more cost-effective. Team chat rooms have proved to be quite useful in real life projects.

16.3.3 Planning and Progress Tracking

The planning of a *sprint* is done in a meeting of the whole team. The *product backlog* features are moved to a *sprint backlog*, divided into tasks and re-estimated. The team can do the estimation using planning poker or other techniques, which for example have been described by Mike Cohn [13]. From our experience planning poker works well in distributed teams as it involves the participation of the whole team, which encourages also active participation of subcontractors. Additionally, in this way of estimating it is easy to notice any differences in opinions concerning the difficulty of a task. When estimates from individual team members differ significantly additional

²<http://www.skype.com>.

³<http://www.lifeseize.com>.

analysis and discussion is needed, as the difference indicates a lack of common understanding.

When the planning is done the tasks are recorded in an issue/task tracking system. In our case JIRA⁴ is used. Issue tracker, like JIRA, allows for following statuses of individual tasks. Finally, JIRA is a web application, which makes easy for all team members to use it regardless of their location.

Practical Tip: Planning poker ensures that also team members from subcontracting organisation take part in the estimates. Additionally, the formula of planning poker, which resembles a card game, relaxes the atmosphere in the team.

In the case of issue status tracking, if the team members do not update regularly the task statuses, it can be done in the daily scrum meetings.

16.3.4 Code Sharing and Development Feedback

A development team must be able to share its work and see feedback on individual work in the context of the whole team. Team members developing a software solution must be able to share their work artifacts in a way supporting versioning. Version control systems allow team members to share code and specify possible access restrictions to it. As the team is spread across multiple sites, all the sites and team members must have an access to the version control. In the case of Solita projects, Subversion (SVN)⁵ is typically used. It is an easy tool to use and can be integrated with many IDEs or file management applications.

In the context of agile distributed teams, techniques of organising version control in a way that supports efficient development can also be implemented. One arrangement of SVN that we use is to have two branches for ongoing development and releasable code. That style of configuring, but in the context of multiple teams, has been described by Henrik Kniberg [14]. This SVN arrangement allows for having a stable code base at any time of the development.

In addition to collaboration on the shared code base, the team must be able to get quick feedback about status of the whole software they develop as a team. Continuous Integration (CI), which was described by Martin Fowler [15], is a technique that ensures that the code submitted to the version control is verified in a build process. The CI tool that we use for Java projects is Bamboo,⁶ but there are also many open source alternatives.

⁴<http://www.atlassian.com/software/jira>.

⁵<http://subversion.tigris.org>.

⁶<http://www.atlassian.com/software/bamboo/>.

Practical Tip: We typically configure the CI tool in a way that all team members get notifications by email if build was unsuccessful. The successfulness of a build can be determined based on compiled code or test results. Usually, also other static analysis are done on the code so that, for example, coding conventions are verified.

16.3.5 Knowledge Sharing

Knowledge sharing among the members of a distributed team is essential for their success. Moreover in this case a tool that can be easily accessed by all team members regardless of their original organisation is most useful. Wiki-like web applications give much freedom in creating and managing documentation on-line. At Solita we primarily use a commercial tool Confluence.⁷ There are many other commercial or free tool alternatives. The main advantage of wiki-like systems is the support for easy page creation, editing, sharing, so that team members can add and edit information themselves. Furthermore, page versioning reveals what kinds of changes have been done, and by whom.

16.3.6 Team Spirit

As a final note about subcontractors in teams, we would like to mention an important aspect of any cooperation, which is a good team spirit. Trust and good relations between all team members must be built up over time, but in distributed teams that aspect of team existence should not be neglected. One channel of providing feedback of the team spirit are the retrospective sessions where all team members can have a say about the project. However, in some cases this may not be enough, particularly if a project is a long lasting one, the team atmosphere can change over time. Therefore, it may be a good idea to create a survey about the team atmosphere. Such a survey can be easily created on-line in free services hosted externally. When the survey is organised this way, the team members can feel comfortable that their responses are anonymous. A survey should have questions covering different aspects of working in a distributed team, starting from technical challenges but also including motivation, identification with the team, workload (too much/too little work) and general feeling about the project work. The results of such a survey should be analysed and if needed corrective actions should be taken.

⁷<http://www.atlassian.com/software/confluence/>.

16.4 Subcontractors and Project Phases

In order to explain all the typical activities in a Scrum project that is distributed and uses subcontractors, we now present an illustrative example project walkthrough. The example project is not any particular project we have done, but rather a representation of a typical scenario, which should be more useful than any particular case. In this walkthrough we point out any issues that in real cases we found important and which may help the readers in their own projects.

This example project walkthrough starts from team assembling, and does not include any activities related to offering the project to the customer. The only selling point which is really relevant to the project organisation, is the project contract model. The most suitable for Scrum projects is a time-material type of contract as it is most natural for this kind of project organisation. On one hand the customer has the freedom to make changes and decide what has to be done, on the other hand the team is not internally limited by certain scope. Additionally, such contract models in a way force more active participation from the customer side, which provides better feedback from the customer. If the project is organised under a fixed scope and price contract, the scope is known, but the implementation order and internal communication of the team still can follow Scrum principles.

We can distinguish the following high level project phases: preparation, development, and release.

16.4.1 Preparation

In the preparation phase the project team is assembled. Knowing the exact or estimated project scope we can predict the resource needs for the project, and also the kind of expertise the project needs. At that phase the subcontractors are selected. Naturally, the initial decision that subcontracting partners can be used is done at the earlier stage and agreed with the customer. Depending on the required expertise a specific partner's resources can be allocated. Our Scrum project typically consists of a team of three to six members. There are no real limits to the size of a team but practically smaller teams may not be feasible, while larger ones may be considered for splitting into smaller groups. If the subcontractors work in a different location than other team members, the sites should be balanced, so that there is not a site with only one developer. In such arrangement the isolation of one developer might impact the team integration and communication needs. If there are at least two or more team members per site, they still can communicate locally and be equally involved, as a site, in the development.

Once the team is known the tools and access to them can be arranged. All the team members, regardless of their location, should have access to the needed tools. The tools depend on the technology of a project, but the communication, knowledge sharing, and tools providing feedback should be accessible across the whole team.

The first activity that the whole team does together is to explain the project scope to the team. It is so called project kick-off meeting. At that point the project practicalities must be explained, too. These include the processes in the project. In the case of subcontractors, especially if they work for the first time with us, it is essential to explain the methodology and ways of working together. If the team is experienced, only the new elements specific to the project must be explained. Additionally, the vision of the project and what is expected from it, must be clear to everybody.

Practical Tip: It is good practice to have the processes and ways of work documented in a shared place, e.g., wiki. So that all the team members can access the documentation when they need to. Moreover as the practices are generally common, they can be reused in major parts across multiple projects.

A project should also collect the basic contact information about all the team members, so that the people could easily find the correct person. Such an internal contact directory may also contain photographs of the people, so that the communication can be more personal even when it is done remotely.

Finally, after assembling the project team, including the subcontractors and internal personnel, the project development phase can start.

16.4.2 Development

The development phase follows the Scrum principles. Therefore the planning session is the first development activity. Such a meeting requires the whole team as well as preferably a customer representative and/or *product owner*. In practice often the *product owner* role is not performed by a person from outside of the team. So the role is taken by the technical lead who knows the background of the project. In some cases it is possible to have a customer representative who is able to present the customer point of view, but may not necessarily be technically capable of recognising dependencies and impact of different features. In that case the *product owner* role is shared between the customer and the technical leader in the project.

During the planning meeting the planned features should be discussed and a few first ones taken into *sprint backlog*, divided into tasks and estimated. This estimation is needed as the team commits to completing specific features in a *sprint*, and the whole team should decide how much time they need. At the project offering stage an initial rough estimation of the features is also done in order to predict possible effort needed for completion of all features. However, the initial estimate is not done by the whole team, it is done by experts planning the project. When the team estimates enough tasks for the following *sprint*, then the commitment for the *sprint* can be done. Additionally, as the efficiency of the team may not be known very thoroughly at the beginning, it is possible to have new features additionally estimated at the planning meeting. These features are not in the scope of a *sprint*, but they can be

taken in, if all other features for which the team commits are done before the end of the *sprint*.

Practical Tip: The distributed team, especially if working together for the first time, should have a chance to meet face-to-face. So the introduction meeting and the first planning meeting can be combined into one session that takes place at one of the development sites. When the team members meet for the first time it also may be worth organising an informal event that brings the team together. A common dinner may be one of the options to start building up the team spirit.

Furthermore in cases when important matters must be discussed a face-to-face meeting including at least the key people, may be more productive, than teleconferences or similar forms of communication.

For the backlogs we use just a spreadsheet which is updated during the planning session. The reason for using a simple spreadsheet is that it is quicker to edit than tasks in an issue tracker, and additionally it can be easily sent to a customer with indicated progress if needed. After the planning the project leader transfers all the tasks into the issue tracking tool.

Then on a daily basis the team discusses the current development status. It is important for the whole team to know beforehand when a feature is considered as complete and can be marked as such in the issue tracking tool. The definition of completion depends on the nature of the project, but for example, for a Java software implementation the definition could include: working code, unit tests, automated acceptance test cases, and documentation. If all these elements that are considered to be mandatory to regard a feature as complete are in place, only then a feature is really complete.

Practical Tip: If the team has problems with following the definition of completeness, individual tasks can be explicitly defined for each element of the definition, e.g., unit tests, documentation, etc. So that the tasks are at a low granularity level. Usually after some time the team members remember that each implemented feature must also include working unit tests and documentation.

For the meetings the teleconferencing tools can be used. If the team notices, and especially the *scrum-master*, that there are issues that must be clarified in order to proceed with the development, then additional workshops on a specific subject can be organised. Such an issue may be a technical impediment, but it can also be a need for finding out customer's preferences for the solution, or some technical decisions that need additional expertise. Furthermore the workshop in practice can be just a short phone call where the issue is discussed.

Practical Tip: If a team is not used to participating in daily Scrum meetings, it may occur that the status reporting becomes chaotic, and that can happen especially when the meeting is organised as a teleconference. A simple solution that we used in such cases, is to have the status records kept in a wiki page that is displayed and edited on a shared screen during the meeting. The status record contains the names of all team members, which determines the order of providing the status, and also the previous status, which allows for observing the changes in status every day. Finally, the use of written status reports, can help to avoid misunderstandings if the voice quality is not very good, as the reported status is visible to everybody.

During the development the team also receives constant feedback about the implementation status from Continuous Integration tool. The builds are done each time the changes are made to the code base and the notification of build failures are sent by email. Moreover all team members can check build statuses in a web UI of the application, which again can be easily accessed by all team members regardless of their location.

A *sprint* ends with a demonstration and retrospective sessions. Practically that can be a single meeting. At the demonstration the completed features are presented. Features can be presented even if the development did not concern any user interface. A running process, web service, file transformations, or database content can be presented. If some features have not been completed (according to the completeness definition), the team should discuss why the features were not completed. Then those features should be re-estimated according to their statuses, and included in the following *sprint*, providing that the features are still within the scope of the project. After the demonstration, the team retrospective can be performed. If the customer actively participates in the development, they can also provide feedback during the retrospective. The team's feedback can be gathered as a *sprint* summary in a wiki page.

Finally, if there are any other features left for the development, then a new planning session is carried out. After that the whole Scrum *sprint* cycle is repeated. If there are any actions based on the retrospective feedback received that should be addressed at the time of new planning, they should be included in the planning. One important aspect of planning is the length of a *sprint*. If the length of the *sprint* seems to be a problem, then it should be adjusted. Typically our *sprint's* length is between two and four weeks.

Practical Tip: In the case of long-lasting projects before a retrospective session a survey about the team spirit can be carried out. Then the results can be discussed at the retrospective session and possible corrective actions can be agreed.

Furthermore at least at the end of each *sprint* the version control system is updated with the latest version of the solution, so that the changes are also trackable at the interim release level in addition to constant version control system updates done during the development.

If all the planned features are done, the development activities may be completed and the project transforms to the release phase.

Practical Tip: We found it useful for team building to have a small celebration after a successful *sprint* completion. As the team is distributed common activities may be difficult to organise, but one way to have a common activity is to have a short team game session on-line, which was actually suggested by one of our customers. The game itself is not as important as a relaxed team activity. It also does not have to take much time, but rather encourage people to share their hobbies and interests.

16.4.3 Release

When the implementation is ready there can still be some final activities for the team. The software to be released should be tested well by the point of the release, however, the customer may require additional tests, or there may be some manual tests that should be executed once more when the software solution is fully completed. If the automated tests cover all the functionality, then naturally manual testing is not needed.

The final activities are the software packaging and possibly deployment. The packaging is always required as the customer must receive a full package containing all ordered deliverables, including code, documentation, test reports. If the customer also requested, the deployment may need assistance from the team side. After the deployment and acceptance tests on the customer side, the project ends, or if any problems are found, then they are fixed and software is delivered again.

The project ends when the software is accepted by the customer. Also then the team gathers for the last meeting to summarise the project. The overall feedback from the customer, team, and the project leader is analysed. Internally the feedback received can be used for the future projects. The team ends the project and the final solution is versioned as the final release. The team members are free to start work in new projects.

16.5 Conclusions

In this chapter we have presented our experiences with subcontracted Scrum teams in a software service company Solita. We discussed different agile practices and

tools highlighting their particular applicability in distributed and subcontracted Scrum teams. We also presented a process of selecting potential subcontracting partners suitable for specific companies, which in our case was a software service company. We presented the process steps and certain selection criteria relevant to our context. We also walked through an example Scrum project that used subcontractors. The example showed when particular agile practices can be used in different project phases. The combination of the selection and later working with subcontractors should provide the reader with a good overview of practices used successfully in Solita.

Additionally, we have shared our research results that can be regarded as indicative data for further comparison with other cases. The data collection and metrics we used, can be developed and adjusted to other organisations' needs. We also noted problems with our current data collection approach, which should be refined over time.

16.5.1 Practical Implications

We have discussed a real-life subcontractor selection process used in a software service company. We also indicated the process aspects (e.g., compatible methodology and culture) that are important during the subcontractor selection in addition to technology and business aspects. We have noted benefits of Scrum feedback loops in the case of distributed projects with subcontractors. We have also gathered a few practical tips that have been used in our projects with subcontractors. The tips included:

- Documentation practices in a way that allows all team members easy access and contribution,
- importance of face-to-face meetings in the process of building team trust and cooperation,
- usage of wiki pages for Scrum meeting minutes to facilitate daily meetings organised as teleconferences, and
- social team activities as an additional way of building team spirit.

These practical tips can be used directly in other Scrum teams, and they can be modified to fit the needs of other environments as well.

16.5.2 Research Implications

Our research findings and detailed data collected from 18 projects of different types, which included 8 Scrum projects, can be used as a reference point in the case of other studies. The presented data contains quantitative data obtained from various sources (e.g., company IT systems and interviews). The data covers a few years between 2006 and 2008, therefore it is relatively extensive for one organisation case.

The subcontracting process as well as the agile tools and practices will be further mastered and developed in our future projects. Therefore, a more detailed and broader study in terms of time and number of analysed projects can be presented in the future. Also we encourage other practitioners to present comparison of findings from their organisations, which would broaden the scope of the research in this area.

16.5.3 Summary

We hope that the presented practices and tools, as well as practical pieces of advice for each phase of a project life-cycle, will be a good source of information for other organisations. Our recommendations can be used by organisations that are about to start cooperation with subcontractors or by those who have already started their cooperation. Naturally, we are not providing a silver bullet recipe for cooperation with subcontractors, but our experiences applied and adjusted to the contexts of other organisation, should at least provide a good reference point for further improvements.

Appendix

Table 16.1 presents detailed findings for 8 Scrum projects and average values for all 18 investigated projects. The other project types included Iterative projects (marked Iter.) and traditional/waterfall projects (marked WF). We have used scale from 1 to 5, where 5 is the best result, for the following metrics: Customer satisfaction, Profitability, and Team performance. In subtotals for different project types, namely ‘Scrum Result’, ‘Iter. Result’, ‘WF Result’, and ‘Grand Total’, the project id (column Proj.) was replaced by the count of projects in given category. Additionally, other values in those last four rows represent average values for the corresponding project type.

Table 16.1 Quantitative results (Abbreviations: Proj. is Project id; Size (mm) is size in man months; Comm. factor is Communication factor; Sub. is Number of subcontractors; Meth. is Methodology used; Cust. satisf. is Customer satisfaction; Profit. is Profitability; Team perf. is Team performance; Cust. invol. is Customer involvement; PM time is Project Manager time)

Proj.	Size (mm)	Comm. factor	Team size	Sub.	Meth.	Cust. satisf.	Profit.	Team perf.	Sites	Cust. invol.	PM time
P03	27	24.00%	6	0	Scrum	4	3	4	2	Full	8.00%
P04	56	20.00%	6	2	Scrum	4	3	4	3	Full	9.00%
P07	19	30.00%	11	2	Scrum	3	4	4	2	Full	7.00%
P09	16	33.00%	4	2	Scrum	1	4	4	2	None	18.00%
P10	29	19.00%	11	2	Scrum	5	5	5	2	Partial	13.00%
P11	7	24.00%	3	1	Scrum	4	4	4	2	Partial	15.00%
P12	10	30.00%	7	5	Scrum	4	5	4	2	Full	14.00%
P18	5	45.00%	3	1	Scrum	5	5	4	2	Partial	16.00%
Scrum											
8	20.96	28.12%	6.38	1.88	Result	3.75	4.13	4.13	2.13		12.42%
Iter.											
4	29.87	22.76%	4.75	0.75	Result	4.5	2.5	3.5	1.75		10.22%
WF											
6	35.84	26.51%	5.83	2.33	Result	3.5	2.83	3.83	2		9.40%
Grand											
18	27.9	26.39%	5.83	1.78	Total	3.83	3.33	3.89	2		10.92%

References

1. Rudzki, J., Systä, T., & Mustonen, K. (2009). Subcontracting processes in software service organisations—an experience report. In Q. Wang, V. Garousi, R. J. Madachy, & D. Pfahl (Eds.), *Lecture notes in computer science: Vol. 5543. ICSP* (pp. 224–235). Berlin: Springer.
2. Rudzki, J., Hammouda, I., & Mikkola, T. (2009). Agile experiences in a software service company. In *SEAA '09. 35th Euromicro conference* (pp. 224–228). Washington: IEEE Computer Society.
3. Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Goal question metric paradigm. In J.J. Marciniak (Ed.), *Encyclopaedia of software engineering* (Vol. 1, pp. 528–532).
4. Bird, C., Nagappan, N., Devanbu, P., Gall, H., & Murphy, B. (2009). Does distributed development affect software quality? An empirical case study of windows vista. In *ICSE '09: Proceedings of the 2009 IEEE 31st international conference on software engineering* (pp. 518–528). Washington: IEEE Computer Society.
5. Siakas, K. V., & Balstrup, B. (2006). Software outsourcing quality achieved by global virtual collaboration. *Software Process: Improvement and Practice*, 11(3), 319–328.
6. Hyder, E. B., Heston, K. M., & Paulk, M. C. (2006). *The esourcing capability model for service providers (escm-sp) v2.01, part 1—the escm-sp-v2: Model overview*. CMU-ITSQC-06-006. Pittsburgh, PA: IT Services Qualification Center, Carnegie Mellon University.
7. Hyder, E. B., Heston, K. M., & Paulk, M. C. (2006). *The esourcing capability model for service providers (escm-sp) v2.01, part 2—the escm-sp-v2: Practice details*. CMU-ITSQC-06-007. Pittsburgh, PA: IT Services Qualification Center, Carnegie Mellon University.
8. Fowler, M. (2006). Using an agile software process with offshore development. Available online. <http://martinfowler.com/articles/agileOffshore.html>. Cited July 2006.
9. Schwaber, K. (1997). Scrum development process. In J. Sutherland et al. (Eds.), *OOPSLA business object design and implementation workshop*. London: Springer.
10. Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *HICSS '07: Proceedings of the 40th annual Hawaii international conference on system sciences* (p. 274a). Washington: IEEE Computer Society.
11. Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2008). Using scrum in a globally distributed project: A case study. *Software Process: Improvement and Practice*, 13(6), 527–544.
12. Herbsleb, J. D., Atkins, D. L., Boyer, D. G., Handel, M., & Finholt, T. A. (2002). Introducing instant messaging and chat in the workplace. In *CHI '02: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 171–178). New York: ACM.
13. Cohn, M. (2005). *Agile estimating and planning*. New York: Prentice Hall.
14. Kniberg, H. (2008). Version control for multiple agile teams. Available online. <http://www.infoq.com/articles/agile-version-control>.
15. Fowler, M. (2006). Continuous integration. Available online. <http://martinfowler.com/articles/continuousIntegration.html>.

Further Reading

16. Eckstein, J. (2004). *Agile software development in the large: Diving into the deep*. Cambridge: Dorset House Publishing Company.
17. Upadrista, V. (2008). *Managing offshore development projects: An agile approach*. Oshawa: Multi-Media Publications.

Part IV

Teams

Chapter 17

Using Scrum Practices in GSD Projects

Maria Paasivaara and Casper Lassenius

Abstract In this chapter we present advice for applying Scrum practices to globally distributed software development projects. The chapter is based on a multiple-case study of four distributed Scrum projects. We discuss the use of distributed daily Scrums, Scrum-of-Scrums, Sprints, Sprint planning meetings, Sprint Demos, Retrospective meetings, and Backlogs. Moreover, we present lessons that distributed Scrum projects can benefit from non-agile globally distributed software development projects: frequent visits and multiple communication modes.

17.1 Introduction

Today, global software development has become a business reality. It offers many potential benefits, e.g., reduced development costs, but also creates significant challenges with respect to communication, coordination, and control. The current dynamic business environment requires projects to work with uncertain requirements and implementation technologies. As a consequence, many software development organizations have started to apply agile development to their geographically distributed projects [1], as agile methods are particularly suitable for projects facing high uncertainty [2]. Due to the physical separation of development teams in distributed projects, many of the key assumptions of agile development, with respect to, e.g., customer interaction, team communication, and being face-to-face [9], do not hold. To gain the benefit from agile development, the practices need to be modified when applied to distributed settings.

Industrial experience reports and a few case studies (e.g. [3, 5–8]) have already shown that agile methods can be successfully customized to distributed projects.

M. Paasivaara (✉) · C. Lassenius
Software Business and Engineering Institute, School of Science and Technology,
Aalto University, P.O. Box 19210, 00076 Aalto, Finland
e-mail: Maria.Paasivaara@tkk.fi

C. Lassenius
e-mail: Casper.Lassenius@tkk.fi

However, the number of reported experiences is still limited and more experiences on how to apply the agile practices to different kind of distributed settings and teams are still needed. The aim of this chapter is to report such experiences collected in four distributed projects.

17.2 Research Methodology

This chapter is based on a multiple-case study [10] of four globally distributed projects that were using Scrum. All the studied projects were developing new software, either a new product, a new service, or a new version of a product. The largest project had seven Scrum teams, all of which were distributed between two sites, while the smallest project consisted of a single distributed Scrum team. One of the projects had two collocated Scrum teams, each at different sites. The development work in all the projects was distributed between two sites, which we call “onsite” and “offsite”. By onsite, we mean the main site, which holds project ownership. In the projects we studied, the onsites were located in Norway and Finland. In addition to project ownership, the onsite locations were responsible for the customer contacts, and most domain and system experts were located at those sites. In each project the offsite location was situated in a country with cheaper labor: Malaysia, the Czech Republic, Russia, and Lithuania. We gathered data from the projects by interviewing project personnel on the practices used, challenges encountered and successes achieved. The semi-structured interviews [4] were recorded and later transcribed by an outside professional transcription company. Altogether we performed 24 interviews each lasting 1.5–2.5 hours. In each project, we interviewed product owners, scrum masters, developers, and testers. We conducted mostly face-to-face interviews with one researcher asking questions and the other one taking notes. Because of our limited traveling budget we could visit only the onsite locations in Finland and Norway, and one offsite location, Czech Republic. However, we were able to interview offsite personnel during their visits to onsites, to Finland and Norway. Moreover, in one case project, we interviewed two offsite team members from Malaysia via SkypeOut calls. Table 17.1 summarizes the case projects and data collection.

In all projects, this was the first Scrum project for the team. The personnel in all teams were very happy about this change from a more traditional process model to Scrum.

17.3 Distributed Daily Scrums

“I think that daily Scrum meetings were the best thing that happened to these distributed teams”—A comment by a distributed Scrum team member.

The daily Scrum meeting is clearly the most important Scrum practice for distributed projects. In this daily team meeting, which lasts approximately 15 minutes,

Table 17.1 Overview of case projects

Case	Type of company & Type of development	Project duration (Scrum usage)	Countries involved (participating personnel)	Teams per site	Interviewees
A	Service company: Development of a new version of printing service software for internal use in new markets.	2 years (5 months)	Onsite: Finland (7) Offsite: Latvia, two sites (2 + 1) Germany (1)	People from all sites viewed as one single team	5: Onsite (4) Offsite (1)
B	Industrial company: Further development and maintenance of an information management tool for internal use.	3 years (14 months)	Onsite: Finland, main site + one subcontractor consultant working close to onsite (9 + 1) Offsite: Russia, subcontractor (6)	One team onsite, one team offsite	7: Onsite (4) + subcontractor consultant working close to onsite Offsite (2)
C	IT company: Further development and maintenance of a large energy software product that is in use in several companies all over the world.	~10 years (1.5 years)	Onsite: Norway (~20) Offsite: Malaysia (~20)	5-7 teams, often combined across sites, number of teams and persons in each team varies across iterations	7: Onsite (4) Offsite (1) face-to-face, 2 over Skype)
D	IT company (same company as in case C): Development of a new intranet for the case company's internal use.	~1 year (7 months)	Onsite 1: Finland (internal customer +3) Onsite 2: Norway (7) Offsite: Czech Republic (4)	People from all sites viewed as one single team	5: Onsite (3) Offsite (2)

each team member answers the three Scrum questions: “What did you do since the last Scrum meeting? Do you have any obstacles? What will you do before the next meeting?” After answering the questions a brief discussion will take place. The purpose of the discussion is not to solve problems, but to decide e.g., who will need to discuss or solve the problems later on.

17.3.1 Application of Daily Scrums to Distributed Projects

When your Scrum project team is distributed between two or several locations, you can arrange distributed daily Scrum meetings, e.g., using videoconferencing, if unavailable, a good quality voice connection will do, perhaps augmented with web cameras, or you can even use only instant messaging. Arranging daily Scrums requires that there is at least some overlapping working time for all participating locations. This is an important consideration when choosing the locations for the project. We think that using asynchronous daily Scrums, e.g. using only e-mail, is unlikely to work well.

Originally, daily Scrum meetings were designed to be arranged as face-to-face gatherings. In a distributed project this is not possible, but you can arrange circumstances that are as close to a face-to-face situation as possible. In the absence of a virtual presence solution, a good quality videoconference connection or even web cameras make it possible to recognize who is talking and to see facial expressions during the meeting. This makes the situation more natural, helps in creating joint understanding and building team spirit for the distributed team.

If your project has several distributed teams that all have their daily Scrum meetings, you can follow the example of one of our case projects. In this project the teams, distributed between two locations, had consecutive daily Scrums. The 15 minute long meetings took place in the same meeting room, one after another. Thus, the connection, with voice and web cameras had to be set up only once.

If video- or teleconferencing is not possible because of technological or other issues, such as problems with spoken language, using chat is an option. Some people might feel more comfortable writing instead of speaking due to (subjective) difficulties with pronunciation or understanding spoken foreign language. One of our case projects used internet relay chat (IRC) for arranging daily Scrum meetings. Typically, all team members wrote their answers to the three Scrum questions prior to the meeting, and the meeting commenced by everybody sending their answers, and reading the others’ messages. Subsequently, a discussion took place. Chat logs were saved for those not being able to participate. When we conducted our last interviews, however, this company had acquired videoconferencing equipment and the team was planning to start using videoconferencing at least once a week for the daily Scrum meetings, thus abandoning their reliance upon text-only daily Scrums. Based upon their experiences, it seems that using only chat for daily Scrums is possible, but we cannot recommend it, as much information is lost compared to tele- and videoconferencing. For example, in a teleconference the tone of voice often

relays important but non-explicit information, as does the facial expressions seen when using virtual presence or videoconferencing systems.

If your project has site-specific Scrum teams they can have normal face-to-face Scrum meetings. However, it is important to share information frequently between the teams at different sites. One possibility is to use Scrum-of-Scrum meetings for sharing information between the teams, as explained later in this chapter. In the case of just a couple of teams, one team member, a representative of the team, can participate in the other Scrum teams' daily Scrum meetings every day or a few times a week to share information between the teams.

17.3.2 Benefits of Daily Scrums

Our interviewees reported that distributed daily Scrum meetings were the most useful practice for distributed projects. The benefits of daily Scrums are numerous: they provide frequent possibilities to share information and coordinate work between distributed team members, they help to recognize possible problems early on, they provide a possibility to create contacts, as well as encourage team members from different sites to communicate more actively, also facilitating off-line communication after the meetings.

Daily Scrum meetings provide a good way for everybody in a distributed team to get an overview of the project situation. In particular, our interviewees reported that it was easier to monitor the offshore situation than before. Moreover, daily Scrums help to identify problems quickly, since with daily monitoring it is difficult to hide problems over a long period of time.

When problems or a need for one-to-one discussion are encountered during daily Scrums, teams should set up separate meetings after the daily Scrums and continue discussions in smaller groups or one-to-one either by video-, or teleconference, chat or email. In all our case projects, daily Scrum meetings encouraged team members to communicate more also outside the meetings, which was seen as one of the greatest benefits of these meetings.

17.3.3 Challenges of Daily Scrums

Even though there are numerous benefits in arranging distributed daily Scrum meetings, there are also some challenges. The biggest challenge for distributed teams is the same as for collocated teams: understanding what the correct amount of information to report in a Daily Scrum meeting is. This is challenging even in a collocated project, but in a distributed project it is even more difficult. Team members do not know what others find interesting or important. Thus, the team needs to practice this with the help of their Scrum master. In one of our case projects, the daily Scrum meetings initially lasted only a few minutes, before the team members learned to

discuss actively, and in particular to be open about their impediments. The Scrum masters started to encourage everybody to talk and share more about their tasks and impediments. Thus, the teams ended up having 15-minute meetings that were found very useful by all participants.

Cultural differences may have a big impact on what people find appropriate in reporting in a daily Scrum meeting. We noticed that there are huge cultural differences in revealing impediments and discussing them in a daily Scrum meeting. For example, in Scandinavian cultures talking about impediments is much more natural than in Asian cultures. Moreover, when team members come from different companies, the risk of team members trying to hide problems is high, in particular in the beginning of a project.

When comparing projects that had distributed daily meetings to projects having mainly site-specific daily meetings, a clear difference could be seen. Most of the participants of the distributed meetings mentioned the benefits: increased transparency to the other site, getting a good overview of what was happening in the project, and well working and open communication across sites. However, the participants of the site-specific, non-distributed meetings mentioned problems: they did not have enough communication and contacts with the other site, nor did they know enough what was happening at the other site. Thus, it is important to share information also between Scrum teams in the same project. Especially when the teams are site-specific, informal communication between the team does not occur naturally e.g., at the coffee table. We will discuss more about sharing information between Scrum teams later on, in the section on Scrum-of-Scrums.

Practical Tips:

- Provide a good infrastructure for daily Scrums. Meetings should be easy to set up and provide as rich communication as possible: virtual reality systems or videoconferencing is best. If unavailable, a good quality voice connection will do, perhaps augmented with web cameras. Use text-only meetings only as a last resort. Avoid asynchronous “meetings”.
- Work actively with the team by practicing and discussing to find the optimum type and amount of information to report in the daily Scrum meetings.
- Create an open atmosphere that makes it easy to raise problems and issues without fear.
- Encourage discussions in small groups or one-on-one after the daily Scrum meetings and arrange a technologically good infrastructure for these distributed discussions.

17.4 Scrum-of-Scrums Meetings

Distributed projects that have two or more Scrum teams need to share information between the teams. One possibility to share information, mentioned earlier, is to

have one team member participating the other teams' daily Scrum meetings. This is practical only when there are no more than two or three teams. When the number of teams increases, arranging Scrum-of-Scrum meetings is a must.

The objective of Scrum-of-Scrum meetings is to share information between teams regarding what is happening in the teams, what kind of challenges the teams are facing, and what kind of interconnections the work done by different teams has. Scrum-of-Scrum meetings provide good possibilities to create contacts and encourage communication between the teams.

One team member from each team participates in the Scrum-of-Scrum meeting as a representative of his or her team. The team decides who participates; the participant does not always have to be the same person. In addition to the team representatives, in one of our case projects all Scrum masters participated in these meetings.

Scrum-of-Scrum meetings normally take place once a week, but they can also be arranged more frequently, if there is more frequent need for coordination between the teams. A suitable length for a weekly Scrum-of-Scrum meeting is half an hour.

During the meeting the three Scrum questions are answered, however at the level of the team. Thus, each team representative tells what his or her team has been doing since the last meeting, what the team is planning to do before the next meeting and what kind of impediments the team has had. Moreover, you can have two additional questions: "Have you put some impediments in the other teams' way?" and "Do you plan to put any impediments in the other teams' way?"

17.4.1 Application of Scrum-of-Scrums to Distributed Projects

Scrum-of-Scrum meetings can be applied to distributed projects in a similar way to daily Scrum meetings. The only difference to collocated projects is the need to arrange the meeting virtually. The same technologies as used in daily Scrum meetings can be used. Good quality videoconferencing provides the possibility to easily recognize who is talking and to see facial expressions. This is important since at least some of the participants may not have met each other face-to-face. In the absence of videoconferencing, web cameras are helpful to support a teleconference call.

17.4.2 Benefits of Scrums-of-Scrums

The Scrum-of-scrums meetings distribute information between the teams and reveal possible problems early on. They open discussion channels between the teams and that way encourage informal communication. One of our case projects that used weekly Scrum-of-Scrum meetings to coordinate actions between their seven Scrum teams felt that these meetings were very beneficial. On the other hand, a project having two site-specific teams at different locations, but not using Scrum-of-Scrum

meetings, mentioned several problems: they did not have enough communication and contacts with the other site, nor did they know enough what was happening at the other site—exactly the problems that Scrum-of-Scrum meetings are designed to prevent!

17.4.3 Challenges of Scrums-of-Scrums

The challenges of arranging distributed Scrum-of-Scrums are the same as for distributed daily Scrums: finding a suitable level of reporting that is both useful and understandable for all parties, cultural challenges and trust issues in reporting impediments, and forwarding the important information to the rest of the team members.

Practical Tips:

- Practice and discuss with the participants about the correct type and amount of information to report in Scrum-of-Scrum meetings.
- Create an open atmosphere that makes it easy to discuss problems and issues.
- Encourage discussions in small groups or one-on-one after the Scrum-of-Scrum meetings and build a good technological infrastructure for such discussions.
- Provide a technically good and easy to set up infrastructure for the Scrum-of-Scrum meetings: a good quality voice connection, preferably also video to be able to recognize who is talking and to see facial expressions.

17.5 Sprints

“Before [we started to use] Scrum I could not really understand when there is a deadline and what should be done by that deadline (...) because there were many different deadlines for customers and development stages (...)”—A comment by a distributed Scrum team member.

Iterations in Scrum are called sprints. The length of one sprint in Scrum is normally from one to four weeks. Our case projects used both four-week and two-week sprints.

If a project has several teams, it is a good idea to synchronize the sprints, i.e., have all teams start and end their sprints at the same time. For example, a large case project had synchronized 4-week sprints in the development teams. The variation of the end and start dates was at maximum a couple of days. The same project also

involved a maintenance team that had a sprint cycle of only two weeks synchronized with other teams' four-week cycle. The reason for this shorter cycle for the maintenance team was to be able to release fixes to customers every two weeks. This system of synchronized four and two week sprints worked well according to our interviewees.

17.5.1 Application of Sprints to Distributed Projects

Using sprints in a distributed project does not differ much from using them in a collocated project. The sprint length in collocated projects can sometimes be as short as one week, but distributed projects make the required meetings more cumbersome, in particular planning, demos and retrospectives, so we think that a two week sprint length is a good minimum sprint length in a distributed project.

The sprint lengths of different site-specific teams should be the same, but there can be exceptions. For example, in one of our case projects the sprint length of the onsite team was four weeks, while the offsite team had sprint duration of only two weeks. The shorter sprint length at offsite made it possible for onsite to better support the offsite team.

Different vacation times in different counties may pose challenges, as team members might be unavailable for a substantial part of a sprint. Sprints might be lengthened to keep sprint content reasonable and the sprints of different teams synchronized.

In the beginning of a distributed project it is beneficial to arrange face-to-face meetings for all team members, so that everybody can at least once meet and learn to know each other. One approach is to invite the whole team to work together in a single location for one or two sprints. This way the team can build a common understanding of the project goals and learn how to work together. Team members also have a chance to get to know each other. After such a collocated period, it is a lot easier to work in a distributed manner. Collocation can be a good idea also when testing and fixing the software, e.g., during the last sprint before a critical release.

One of our case projects used collocated sprints on a need basis. In this project, especially team members from offsite travelled to the onsite location. The visits normally lasted between two and four weeks, which made it possible for team members to really work together. In particular during critical phases, it was considered important to collocate the team, e.g. for the last sprint before a release or for the first sprint in a new release project, when most of the planning took place.

17.5.2 Benefits of Sprints

Short sprints hugely increase the transparency of distributed projects. Sprints with clear deadlines and goals, make it easier for all team members to understand what

is supposed to be done during the next sprint. In particular, team members at offsite locations benefit a lot, since often the offsite team members do not have a clear picture of the overall project in a traditional distributed setting.

In addition, the frequency of feedback between onsite and offsite is increased. There is no possibility to delay the completion of a task because it is “only 95% ready”. Moreover, short sprints reveal quickly, e.g., if offsite personnel have misunderstood the requirements, and the problem can be solved immediately.

Finally, short sprints, with frequent regular meetings, make it easier for a distributed team to create a joint team identity, making members feel like they are on a single team working for common goals, rather than being on two or several separate teams, not really understanding each other.

17.5.3 Challenges of Sprints

The main risk with planning on a sprint by sprint basis is losing track of the “big picture”, i.e. the overall goal of the project. Keeping the overall goal in mind is important, not least from the point of view of the resulting product architecture.

The planning overhead, involved in each sprint, can tempt one to use too long sprints, in which case their benefits erode.

Practical Tips:

- Synchronize sprints between teams
- Do not have sprints that are shorter than two weeks in a distributed project
- Arrange collocated sprints when starting a project or facing challenges so that the whole distributed team can work together

17.6 Sprint Planning Meetings

At the beginning of each sprint, teams hold a sprint planning meeting. In the meeting, the backlog items to be developed in the sprint are selected, broken down to tasks, and their effort is estimated. The product owner presents and explains to the team the backlog items and answers the team members’ questions. Then the team plans the sprint together.

17.6.1 Application of Sprint Planning Meetings to Distributed Projects

If a Scrum team is distributed, the sprint planning sessions can be arranged as distributed meetings.

However, if possible, it can be a good idea to invite all distributed team members to a single location to plan the next sprint face-to-face. If team members are not too far apart, this can be arranged regularly. You can also consider arranging collocated sprint planning meetings at least for the first or first few sprints. That makes it possible for the team members to meet at least once face-to-face, to get to know each other. Unfortunately, in most cases arranging face-to-face meetings in two or four week intervals is not economically feasible.

In one of our case projects, where the onsite and offsite locations were located at a reasonable distance—a one-hour plane trip from each other—a couple of offsite team members flew to the onsite location for the first few sprint planning meetings, which made these meetings more efficient. After this good start, the meetings were arranged in a distributed manner supported by teleconference and application sharing.

Another possibility is to divide the sprint planning meeting into parts: collocated meetings at the different sites and a common distributed meeting for the whole team. In a large case project, with only three hours synchronous working time between the onsite and offsite locations, the sprint planning meetings were divided into three parts: a distributed meeting, a local meeting at onsite, and a local meeting at offsite. The distributed meeting was arranged using teleconferencing and application sharing. During the distributed part, the product owner presented the prioritized items in the backlog, and the team asked questions. Because of the time-zone difference this part of the meeting was time-boxed for the three common working hours for both sites. After the meeting, the offsite working day ended. The onsite team continued by dividing the backlog items into more detailed tasks, adjusting the estimates made by the product owner and making initial assignments of the tasks to different team members. The offsite team continued the work the following morning by discussing and commenting on the draft plan they had received from onsite.

If a project has site-specific collocated teams, sprint planning meetings can often be arranged face-to-face. However, even if the team is collocated, the product owner may be located at another site, introducing the need to arrange at least a part of the meeting virtually.

17.6.2 Benefits of Sprint Planning Meetings

Sprint planning meetings provide team members an opportunity to participate in planning and thus both better understand what is expected of them, and to commit to the plans. In a distributed team, these meetings provide visibility to the work on both sites and offer a regular discussion forum. Sprint planning meetings also provide opportunities for building team cohesion and identification, despite of distribution.

17.6.3 Challenges of Sprint Planning Meetings

Arranging a distributed meeting is always a challenge. All our case projects found collocated planning meetings preferable. However, despite the fact that all case projects had positive experiences with collocated meetings, arranging them regularly proved too expensive. Project members commented that planning is a challenging task that requires lots of discussion, which is difficult to do efficiently while distributed. Also, the issues discussed are sometimes just difficult to explain when distributed.

Long distributed meetings can be also very tiring, if, e.g. the voice connection between the sites is not very good—a situation that was not uncommon to the team members we interviewed. Moreover, if videoconferencing or web cameras are not used, it can be difficult to know who is talking when not seeing the persons from the other site.

You have to take into account cultural, as well as knowledge differences between the sites. Otherwise, experienced developers at onsite might end up doing the planning with offsite developers just listening and not actively participating.

Practical Tips:

- If possible, plan visits so that sprint planning meetings can at least sometimes be arranged face-to-face
- Encourage all team members to participate actively in planning
- Ensure technically good circumstances for virtual meetings: a quality voice connection, a working video connection if possible, and application sharing

17.7 Sprint Demos

At the end of a sprint, the team demonstrates the developed functionality to all interested parties. The meeting is called a sprint demo or sprint review meeting.

17.7.1 Application of Sprint Demos to Distributed Projects

In a distributed project, demos are normally arranged in a distributed manner. Even though the project team might be collocated, there are often parties, such as the product owner or team members from other teams, who are interested in participating in demos from other sites. If your project has just a couple of teams, they can have joint demos. This allows the teams to give and receive immediate feedback.

All our case projects arranged demos that both onsite and offsite personnel participated in. The demos were normally arranged using teleconference and application sharing. During visits, face-to-face demos were sometimes arranged.

17.7.2 Benefits of Sprint Demos

The demos increase the visibility of the project to all participants of the demo, especially between the distributed sites. They also offer a possibility to give and receive feedback, as well as to monitor the work at offsite. For example, in one of our case projects, before starting to use Scrum, the onsite and offsite teams were working independently for long periods of time. This commonly led to a lot of rework for the offsite team, as they often misunderstood the requirements written by the onsite personnel. Short sprints with demos at the end mitigated this problem.

17.7.3 Challenges of Sprint Demos

The biggest problem with demos is often the same as with other distributed meetings: the technology does not offer good enough possibilities to communicate efficiently. Our case projects typically used teleconferencing with application sharing to arrange their demos, but they were not happy with this technology.

Practical Tips:

- In a multi-team project, invite also members from other teams to the demo and provide for a possibility to participate in a demo also virtually, since even though the team might be collocated there can be interested parties from other sites. A demo provides a good possibility to share information, and to give and receive feedback
- Ensure technically good circumstances for virtual meetings: a high quality voice connection, video connection if possible, and application sharing

17.8 Retrospective Meetings

A retrospective meeting normally takes place at the end of a sprint. During that meeting the team discusses three questions: “What has been good during this sprint?”, “What has not been that good?” and “What kind of improvements could we do?”

17.8.1 Application of Retrospective Meetings to Distributed Projects

Retrospectives in distributed Scrum projects can be arranged in similar ways as the planning meetings. In particular in retrospectives, it is important to create an open

atmosphere in which everybody's input is welcome and valued. In particular, people at the onsite location should try not to dominate the meeting too much.

Technically, the meeting can be done using tele- or videoconferencing, and perhaps application sharing to jointly write the minutes.

In two of our case projects, the retrospective meetings took place directly after the demos as distributed teleconference meetings.

17.8.2 Benefits of Retrospective Meetings

The main benefit of the retrospective meeting is that it provides a time in which the whole team reflects upon its own behavior, and how to improve it. Conducted successfully, the meeting can also provide good opportunities for increased team identification and commitment.

17.8.3 Challenges of Retrospective Meetings

In order for retrospectives to be successful, it is important that everybody participates and tries to contribute. Dominant personalities and experts should make a special point of recognizing contributions from distance members. As an example of a suboptimal practice, one case project with several Scrum teams had retrospectives that consisted only of the Scrum masters. While this can be useful as such, it is no substitute for retrospectives that involves the whole team.

Practical Tips:

- Make sure that the whole team participates actively in the retrospective meetings
- Create a positive and open atmosphere that makes it easy to participate; recognize even small contributions
- Be sure to follow up on the issues raised and suggestions presented in the meeting

17.9 Backlogs

Backlogs are lists of items, e.g., features, to be developed. In the sprint planning meeting, the product owner, with his or her team, selects items from the product backlog to be developed during the next sprint. The features are then broken down into tasks that are estimated and placed in the sprint backlog. There are several commercial and open source tools for managing backlogs.

17.9.1 Application of Backlogs to Distributed Projects

Collocated Scrum teams may manage their backlogs using physical objects, such as post-it notes on the wall. In a distributed team, this is not practical, since all team members, as well as their product owner, need to get access to the backlog. Thus, electronic tools are needed.

Our case projects used different tools to manage their backlogs, e.g. Wiki was used by a small project, whereas Jira was used by a large project. There are also specific backlog management tools available, such as Scrumworks and an open source tool Agilefant, but none of our case projects used any of them.

How the backlog is managed needs to be decided together with the team, and the responsibilities for backlog management clearly assigned. In particular, the responsibilities of the product owner are critical.

17.9.2 Benefits of Backlogs

Electronic backlogs with up-to-date information and access by all team members are necessary for managing tasks and monitoring progress in a distributed Scrum project.

17.9.3 Challenges of Backlogs

The biggest challenges related to the backlogs in our case projects were related to unclear updating responsibilities. Especially the responsibilities of the product owner were quite unclear for new people assuming that role. In some projects, also the Scrum masters, chief designers or chief architects performed some of the product owner's responsibilities. Moreover, in some projects there were several product owners who had not clearly divided responsibilities between them.

Practical Tips:

- Choose a tool suitable for your purposes and give access to all team members
- Agree on updating responsibilities

17.10 Frequent Visits

In addition to learning how to apply Scrum practices to distributed projects, the distributed Scrum teams need to take into account lessons learned from managing

distributed software development projects in general. One important lesson is to arrange visits for distributed team members frequently enough.

17.10.1 First Visit

Building an efficiently working and communicating team is less painful if the team members can meet each other face-to-face at least in the beginning of the project. Preferably, the first visit should not be only a short trip to meetings, but a longer stay during which distributed team members can start working together on project tasks. The length of a collocated working period could be, for example, one or two sprints. During this face-to-face period the team members learn to know each other and develop joint working habits by working together at least for a short period of time. This is an efficient start-up for a project and makes it easier to communicate and collaborate later on when team members are working from different sites.

17.10.2 Further Visits

Later on during the project, both short trips and collocated working periods are useful. A collocated working period can be scheduled, for example, for a critical project phase like the last sprint before a release or for the first sprint of a new release, when most of the planning takes place. It is a good idea to schedule the short trips so that the visitors can participate the regular meetings face-to-face, making the meetings more efficient. Thus, it is ideal to schedule trips at the end of a sprint, so that the visitors can participate in the sprint demo, retrospective meeting and sprint planning meeting for the next sprint during the same trip.

In the beginning, when arranging the first meeting or a collocated working period for a new team, it is important that the whole team can participate. Later on, when arranging collocated working periods or short trips, the whole team does not necessarily need to travel. Instead, a few team members at a time can spend time at a remote site, on a need basis. However, it is important that it is not always the same persons that travel, but that every team member gets his or her turn. You can, for example, create a travelling schedule for the project. Moreover, it is a good idea to arrange trips to all sites, so that team members will get to know the circumstances at different sites and at the same time learn more about the local culture of their teammates. This way the task of travelling can be divided more evenly between the team members.

When planning a travelling schedule for your project, you can plan a regular schedule, for example a short trip every second sprint, or base your plan on the critical phases of your project schedule. Moreover, you probably have to arrange trips on a need basis. When your project is facing challenges, they are often easiest to solve face-to-face.

All our case projects arranged visits between the sites for team members, either on a need basis, or according to a regular schedule. The visits also included leisure activities, such as sauna or dinner. These gave team members a good possibility to get to know each other on a personal level.

17.10.3 Benefits of Frequent Visits

Frequent visits provide good opportunities for getting to know persons from the other sites, discuss difficult issues, and get a better picture of the project. Face-to-face meetings also increase trust between team members and encourage them to continue communication after the visits. It is important to arrange visits not only in the beginning of the project, but also during the project. All our case projects found their current model of frequent visits as very useful and even more visits were hoped for.

17.10.4 Challenges of Frequent Visits

Travelling comes with a high cost both in working time and money, thus it is important to plan the trips carefully. Getting travel plans accepted by higher-level managers is often thought to be the most challenging part of frequent visits. Motivating the need to travel to managers who might not appreciate the importance of meeting face-to-face and working together to build the team can be difficult. Explaining that a trip will pay itself back quickly in better communication and in more efficient teamwork might not do it, since management might expect you to do perfectly well without the team having a chance to meet face-to-face! However, in our case projects this problem was never mentioned. Even though the case projects arranged quite a few trips, none of the interviewees mentioned any problems of arranging trips due to cost.

Finding time to travel can be more difficult, especially for experts who have more than enough to do anyway. In our case projects, mainly offsite personnel did the travelling. The reason for this was that onsite personnel was mainly experts who did not have time to travel, even though that was hoped for by the people at the offsite locations. Offsite personnel were mainly developers, who found it extremely useful to meet the onsite experts face-to-face and ask questions and discuss difficult issues. Our interviewees, especially from offsite, hoped that onsite personnel would travel more to provide opportunities for additional offsite persons to meet them and to share the sometimes quite heavy and tiring traveling duties between onsite and offsite.

Finally, one challenge related to frequent visits, especially in arranging collocated working periods is limited office space. Quite often this problem can be solved by planning ahead, e.g. by reserving a big enough team room or reserving a meeting room for the time of a collocated period.

Practical Tips:

- Even though you might have a good infrastructure for electronic communication, face-to-face meetings are needed to build a common understanding and an efficiently working and communicating team
- Start a project preferably by a collocated sprint
- Plan the travelling schedule in the beginning and remember travelling costs in your budget
- Divide traveling responsibilities between your team members and sites

17.11 Multiple Communication Modes

In addition to face-to-face discussions, members of distributed Scrum teams need to communicate a lot electronically. Providing several good tools for different kinds of communication purposes is another lesson learned from managing distributed software development projects.

Different people, contexts and situations require different communication tools. The minimum set of tools that should be provided include:

- email
- instant messaging
- unrestricted voice calls
- application sharing

In addition, web- and videoconference solutions should be made available if possible. Often, videoconferencing equipment is a scarce resource. Optimally, one videoconference or telepresence room could be made available for spontaneous short meetings only, e.g. it cannot be reserved or used for hours by a single meeting.

In our case projects, tool choice seemed to depend both on the purpose of the communication, e.g. chat was used to ask short questions or for checking whether the other party was available to receive a phone call, as well as the preferences of a user. Some people preferred synchronous voice communication, while others with limited language skills preferred written communication.

17.11.1 Benefits of Multiple Communication Modes

The main benefit of allowing and providing for multiple communication modes is that it lowers the barriers to communication by allowing team members to communicate in a way that fits them the best outside project meetings. Since poor communication or the lack of communication altogether is a common problem in distributed projects, one should not underestimate the importance of this.

17.11.2 Challenges of Multiple Communication Modes

Providing for multiple ways of communicating is in principle easy, but corporate policies and IT departments sometimes make it unnecessary difficult. Try to find a way of providing for, in addition to email, at least the possibility for instant messaging and unrestricted voice communication between team members. For voice, an IP-based solution can help mitigate the fear of otherwise high phone costs.

Practical Tips:

- Aim at providing a rich set of communication tools that personnel can use also outside official meetings
- Allow people to use the media they like the best with the least possible limitations. For traceability or other reasons, documentation can be done after an informal exchange, e.g. by email

17.12 Conclusions

In this chapter we have discussed the use of Scrum practices in global software engineering, as well as provided practical tips for how to apply them. We discussed the use of distributed daily Scrums, Scrum-of-Scrums, Sprints, Sprint planning meetings, Sprint Demos, Retrospective meetings, and Backlogs. In addition, we discussed overall lessons learned in global software engineering that can benefit distributed Scrum projects.

References

1. Ågerfalk, P., & Fitzgerald, B. (2006). Introduction. *Communications of the ACM*, 49(10), 26–34.
2. Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*, 34(11), 131–133.
3. Fowler, M. (2006). Using an agile software process with offshore development. <http://martinfowler.com/artcles/agileOffshore.html>. Referenced: 19.12.2007.
4. Patton, M. Q. (1990). *Qualitative research and evaluation methods*. Newbury Park: Sage Publications.
5. Simons, M. (2002). Internationally agile. *InformIT*, March 15th.
6. Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *Proceedings of HICSS 2007* (p. 274a).
7. Sutherland, J., Schoonheim, G., Rustenburg, E., & Rijk, M. (2008). Fully distributed scrum: The secret sauce for hyperproductive offshored development teams. In *Proceedings of agile conference, 2008. AGILE '08* (pp. 339–344).

8. Sutherland, J., Schoonheim, G., & Rijk, M. (2009). Fully distributed scrum: Replicating local productivity and quality with offshore teams. In *Proceedings of HICSS 2009* (pp. 1–8).
9. Turk, D., France, R., & Rumpe, B. (2005). Assumptions underlying agile software-development processes. *Journal of Database Management*, *16*(4), 62–87.
10. Yin, R. K. (1994). *Case study research, designs and methods*. Thousand Oaks: Sage Publications.

Chapter 18

Feature Teams—Distributed and Dispersed

Jutta Eckstein

Abstract Teams have to be enabled for delivering business value to customers. Organizing project members in feature teams provides the basis for doing so. Some (large) global projects are organized in distributed feature teams, where each feature team is co-located at one site and some in dispersed feature teams, where feature team members reside at different sites. Besides focusing on delivering business value projects have to ensure conceptual integrity of the system. While feature teams deliver the business value, a technical service team ensures conceptual integrity (e.g. adherence to the same look-and-feel) across the whole system.

18.1 Introduction

The core motivation for agile development is to provide, at any point in time, the highest possible business value for the customers in terms of working software. This is a challenge even for a co-located team and it is increasingly difficult the more distributed a team is. The team structure of a (large) global project can hinder or support this goal. A project is defined as large and global if it is distributed over more than one site and more than one team is—or in other words more than fifteen developers are—working on it.

If a global project needs more project members than can successfully work together in a single team—which is typically more than ten people—the whole project team should be divided into subteams.

The first section of this chapter explains the historical organization of these subteams which has its origin in following a linear or rather waterfall process. The next section elaborates how structuring a project in feature teams enables agility. Moreover it discusses if these diverse feature teams reside at one location (distributed teams but each feature team is co-located) or span multiple locations (dispersed

J. Eckstein (✉)
Gausstr. 29, 38106 Braunschweig, Germany
e-mail: feedback@distributed-teams.com

teams). The implications and specifics of these different settings are elaborated in detail.

Organizing the whole project in feature teams supports the concentration on the business value. Yet, it is very likely that different technical concepts emerge, because different feature teams will for example develop different access layers to the database. The third section explains how a technical service team helps avoiding these discrepancies from happening and ensures instead conceptual integrity by serving the feature teams.

18.2 Context

The experiences described in this chapter refer to several projects I have been working on. Moreover, I verified some of my experiences through frequent exchanges with colleagues of mine. Please find below some ranges for the characteristics of these projects:

- Most of my projects are rather large in terms of people, between thirty and 300 project members.
- The smallest distribution degree is two sites within one country and the biggest one six sites spread over the globe.
- The following countries were involved: Austria, China, Czech Republic, Germany, Hungary, Poland, Singapore, Switzerland, UK, USA.
- The domain for the system varied very much, we built systems for embedded products, financial applications, mechanical engineering, multimedia, and telecommunication.
- Although I refer to all of them as “projects” this doesn’t cover the truth, because some of them are actually product development. The biggest difference between the two is that product development has no ending—as soon as the first version is shipped this product is maintained and additionally the team works on the next version of the product.
- Some of these “projects” were greenfield applications whereas other ones were existing ones with a lot of legacy.

I hope this context description helps in order to classify and better understand the experiences described and the recommendations given.

18.3 Historical Structures of Distributed Teams

The reduction of communication between sites is historically the major motivation for the decision on team-organization in global projects. In such a situation the team structure is often based on the different phases, activities, or roles found in linear development. For example, testers may be located at one site, business analysts at the next and designers at an additional site. Or as Shao and Smith David wrote, when describing global development, that using a waterfall approach:

[...] implies that front-end activities such as preliminary requirement analysis and conceptual architecture design as well as back-end tasks like system testing, system deployment and user training will remain in place. [1]

Thus, according to Shao and Smith David, each of the waterfall phases is conducted at one location and moreover typically the front-end and back-end activities are staying at the headquarters whereas the middle activities like coding are transferred to a different location. The subteams are shaped according to activities or rather phases. Building co-located subteams this way makes collaboration difficult. Distributing these activities across different sites makes it even harder.

Another popular team structure, which is often combined with a structure along activities, is one which uses the technological know-how of developers, or rather the architectural layers of the software, as its guiding principle for defining team boundaries. In the resulting structure some people who concentrate on user interfaces reside at one site, database specialists at the next, and middleware experts at a third site.

18.3.1 Consequences

Often the structure along activities is combined with the one along technical components. Such a combinational structure could result in a project made up of a team of business analysts in the USA, of user interface specialists in Northern Ireland, and of testers in China. Given such a structure, it is not surprising how often the following consequences can be observed:

- Things, in general, do not fit together. For example component interfaces often suffer from compatibility issues.
- Developed functionality does not serve the customer.
- Blaming between sites becomes the norm because nobody has the full responsibility of delivering a feature. Every team or site comprehends a partial responsibility of the delivery only.

Team structures following either activities or/and technical components are the reason why it is often so difficult to deliver business value during the project's lifetime (and also at the deadline of the project). This is particularly unfortunate because only the early delivery of business functionality can trigger the valuable customer feedback. Without such feedback it is not possible to learn from the customer and allow the system to gradually adapt to the users' needs.

18.4 Building Agile Teams

In order to always keep the business value of your customer in mind, and enable a (sub-) team to deliver business features there is only one solution: organize teams along features. This is also an implicit request of one of the principles of the agile manifesto:

The best architectures, requirements, and designs emerge from self-organizing teams.

That is, instead of structuring teams according to technical know-how or activities, organize teams according to business domain areas. A single feature team should always be able to deliver whole business functionality (features or stories) or in other words business functionality should never be split across several teams. Taking the whole responsibility for features allows a team to organize itself and its work.

This requires every feature team to assemble all roles, knowledge, and skills (also to acquire the missing knowledge) that are necessary to deliver a complete business feature. Consequently, a feature team will consist of analysts, testers, user interface specialists, database experts, and so on—just everyone who is needed for a complete delivery of the required functionality. Feature team members might be experts in their specific field however they have to take turns in fulfilling different roles. Only the latter allows the feature team to become a generalist in its business domain and makes it possible to spread the knowledge across all its members. This in turn reduces the risk of depending on specific experts and creates an environment where all feature team members can support each other.

18.4.1 Feature Teams—Co-located or Dispersed

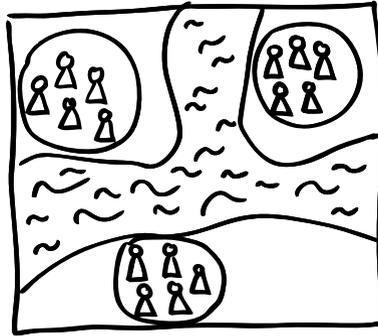
Structuring teams according to features or domain areas requires creating multi-disciplinary subteams. Yet, how can you possibly organize teams across different sites and still ensure this concept? Generally there are two possibilities:

- **Distributed feature teams.** The first option, especially if you want to simplify the communication within the team, is to create feature teams site-wise. Thus the project will be structured in multiple feature teams where each feature team will possibly reside at a different location, yet the team members of every feature team are co-located. As prerequisite for such a setting all roles, knowledge, and skills (also to acquire the missing knowledge) about a feature team's domain area exists at the feature team's site. However, this may not always be the case. To resolve this you can either ask the people who have the required know-how to transfer that missing knowledge to the specific site, or to ask those people to move to that site and keep the feature team physically together this way. Alternatively implement the subsequently described strategy of dispersed feature teams.

For distributed feature teams you have to take into account that although the feature team's internal communication is easier because the team is co-located, the communication across the different feature teams is often harder because of the physical distance between the sites where the feature teams are situated. A working communication across teams can be critical for the project, because it will be needed to enable—among other things—conceptual integrity.¹

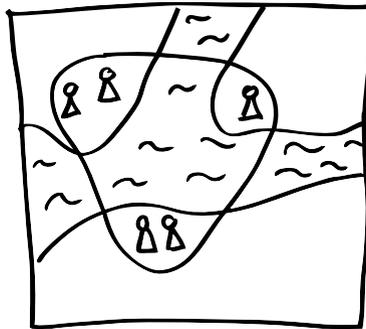
¹Conceptual integrity refers to applying the same concepts, such as e.g. the same look and feel, across the whole system.

Fig. 18.1 Three distributed feature teams working on one project (or product)



- Dispersed feature team.** If the roles, knowledge, and skills are spread over many sites and it seems impossible to co-locate all people comprehending the necessary wisdom, you have to consider a different approach for building feature teams: Consciously establish feature teams across different sites. Thus such a multi-site feature team will be distributed in itself with members being dispersed over diverse sites. Internal team communication will require a higher effort. Having a joint objective becomes more important for dispersed feature teams in order to jell as a team. Yet, ensuring the delivery of features iteration for iteration provides such a joint objective. Of course, it is not quite as easy. For this to work well, the team needs to share some common ideas, such as appreciating the same set of values and work ethic. This is discussed further in the next section.

Fig. 18.2 One dispersed feature team working on one project (or product)

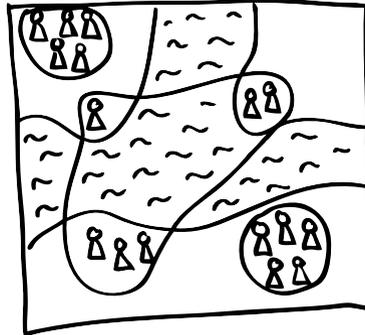


Having dispersed feature teams in place eases this cross-team communication by the physical proximity of individual team members to people belonging to other feature teams. This is based on the fact that members of different dispersed feature teams are co-located at the same site.

- Distributed as well as dispersed.** As we have seen a (large) global project has actually two possibilities for structuring teams. Each possibility has advantages and disadvantages. In most of my projects we establish a mix—some of the teams are co-located and thus distributed feature teams, others are dispersed ones. The

structure for the feature teams should always be driven by the business domain that needs to be implemented. We typically take the business domain together with the required technological know-how as the basis for the team structure. Thus for the sites where we have both—the required business and the technical knowledge available—we create a co-located feature team. For the business domains (or/and for the technology) where we don't have all the requested knowledge available at a single site we create dispersed teams.

Fig. 18.3 Two distributed feature teams together with one dispersed feature team working on one project (or product)



18.4.2 *Creating Proximity for Dispersed Feature Teams*

A team is more than the assembly of individuals. The individuals or rather the team members should trust and support each other and pull together towards a joint goal. Tuckman has described in his team psychosocial development model how a team develops trust. This model explains that every team, in order to become a team has to go through different stages. These stages are forming (getting together, defining objectives), storming (first conflicts with consequences on roles), norming (accepting rules and evolving guidelines), and performing (working towards a common goal, constructive resolution of conflicts) [2].

In order to go through these stages timely, the members of the dispersed team ideally work together (that is by being co-located) at the beginning for a limited timeframe. Being co-located is not a natural setting for members of a dispersed team, yet it will still speed up the process of going through the different stages. Although taking more time, starting with a dispersed team in its *natural* (dispersed) environment makes the most critical challenge of their natural environment transparent quickly: Creating proximity over the physical distance. Depending on the degree of the team's dispersion there are different options for creating this proximity. If the team spreads for example over Central and Eastern Europe, then team members can meet at one of the feature team's site for a few days every week or at least for every

iteration turnover.² If the distance is much bigger you should consider the concept of expatriates, where team members work at a different team location for a longer period of time. Imagine you have a feature team composed of five Russian developers and two German developers. A good way to help the team jell is to have the two Germans work for the start of the project as expatriates in Russia.

It is crucial to acknowledge that face-to-face meetings are most effective when it comes to creating solidarity and intimacy among team members. This is also emphasized by Vicki R. McKinney and Mary M. Whiteside who conducted a survey with more than 200 individuals working in virtual teams, and, one of their conclusions is

[a] prior traditional relationship is a shared characteristic of many distributed relationships. [3]

18.5 Technical Service Team Ensures Conceptual Integrity

Having feature teams in place will ensure the focus on delivering the highest business value. Without an architect, it might happen that the feature teams focus on business features only and not on conceptual integrity. So you might end up with a system showing different look-and-feels, using diverse possibilities for database access and the like. Only adherence to conceptual integrity makes it possible to understand and maintain the system easily, because the same concepts will be applied everywhere. That is the reason why conceptual integrity is the basis for simplicity and for maintainable systems. The agile manifesto asks for simplicity in the following principle:

Simplicity—the art of maximizing the amount of work not done—is essential.

In many projects it is the architect who takes care for conceptual integrity. Depending on the size of your project as well as on the complexity of it, this one architect might need the support of additional architects.³ For ensuring conceptual integrity, some systems require a specific infrastructure or a framework. For example if your system is based on a distinct middleware, embedding and using the middleware should follow the same approach in the whole system. Another example is product line development where different products are based on the same architecture. In such settings a so-called *technical service team* will develop for example, an architecture or a framework providing the necessary services for all feature teams. The feature teams will in turn build their features on top of these services using the same concepts.

The technical service team provides services requested by the feature teams and understands the feature teams as their customers. The technical service team differs from the classical team of architects sitting on an ivory tower and creating frameworks nobody can and wants to use.

²Iteration turnover is the ending of one iteration with review and retrospective and the planning of the next iteration.

³More on the role of the architect in *Roles and Responsibilities*, see Chapter 19.

For the technical service team to act as a pure service provider, the feature teams have to accept their role as customers. In order to do so, they need to assign somebody as the product owner for the technical service team. This product owner, just as a “regular” one, decides on priorities and steers the development of the technical service team. Compared to the customers of feature teams, the customers of a technical service team are always developers (but in a sense also end users—of the technology provided). And compared to the business features the feature teams are developing, the technical service team works on technical “features”. Yet, the feature teams will ensure that the technical features are business-driven, because the feature teams will always request what they need in order to develop business functionality.

18.5.1 Starting Team as Role Model

It is very rare that product development begins with—for example—a hundred developers organized in ten to fifteen subteams on day one. Instead, most large global projects are hardly started with more than one team. This one starting team should have the task of implementing two to three key user stories together with a referential architecture. Depending on the complexity of the system under development, this first referential architecture might be sufficient to ensure conceptual integrity. The subteams joining the project later can use this initial implementation as a role model for their further development. This is actually also the reason why this initial architecture is called referential architecture—it serves as a reference later on (which doesn’t mean it can’t be changed).

A reasonable creation of the starting team takes people from all project sites into account. This setting ensures that all sites will create some knowledge about the referential architecture already during the starting phase. This in turn enables knowledge-spreading about conceptual integrity across sites.

18.6 Conclusions

It is crucial to *enable* a team to deliver a whole feature. By having an entire team responsible for whole business functionalities nobody can be blamed if, at the end of an iteration, not all tasks have been completed. Instead a feature team has to pull together and make the features happen.

Organizing a (large) global team in subteams requires deciding on distributed or dispersed feature teams. Team members of a distributed feature team will be co-located whereas the ones of a dispersed feature team will be spread over multiple sites. A joint goal, like regular feature delivery, helps every team to jell. A dispersed team will additionally need some time to create a team identity so they work together more effectively.

Cross-site communication should be in the focus when establishing distributed (co-located) feature teams. Cross-site communication ensures the common understanding of the system and the joint objective of the entire project.

While feature teams ensure the focus on the delivery of business features, they might lose sight of conceptual integrity. Technical coherence of the system can be provided as a service by a technical service team. The basis for conceptual integrity is provided by the starting team.

References

1. Shao, B. B. M., & Smith, D. J. (2007). The impact of offshore outsourcing on IT workers in developed countries. *Communications of the ACM*, 50(2), 89–94.
2. Tuckman, B. (1965). Developmental sequence in small groups. *Psychological Bulletin*, 63, 384–389.
3. McKinney, V. R., & Whiteside, M. M. (2006). Maintaining distributed relationships. *Communications of the ACM*, 49(3), 82–86 (Quote on p. 85).

Further Reading

4. <http://www.agilemanifesto.org>.
5. Eckstein, J. (2004). *Agile software development in the large*. Cambridge: Dorset House.
6. Eckstein, J. (2010). *Agile software development with distributed teams*. Cambridge: Dorset House.

Chapter 19

Roles and Responsibilities in Feature Teams

Jutta Eckstein

Abstract Agile development requires self-organizing teams. The set-up of a (feature) team has to enable self-organization. Special care has to be taken if the project is not only distributed, but also large and more than one feature team is involved. Every feature team needs in such a setting a product owner who ensures the continuous focus on business delivery. The product owners collaborate by working together in a virtual team. Each feature team is supported by a coach who ensures not only the agile process of the individual feature team but also across all feature teams. An architect (or if necessary a team of architects) takes care that the system is technically sound. Contrariwise to small co-located projects, large global projects require a project manager who deals with—among other things—internal and especially external politics.

19.1 Introduction

If the world would be ideal, then this chapter would be superfluous. Because then everyone working on a system would do the right thing. For small co-located agile teams software development often comes close to the ideal case. Yet, unfortunately this isn't the case for large global projects—projects that are spread over at least two sites and consist of more than one team or more than fifteen developers. In the latter setting various roles and responsibilities have to be implemented for successful collaboration.

The first section talks about the more classic roles, like database expert or user interface designer and what happens with those roles in an agile feature team. The next section elaborates on the role of the product owner. This person should steer a feature team business-wise. In a global project this means that the product owner needs to be close to both—the feature team and the customer. And moreover, in a

J. Eckstein (✉)
Gaussstr. 29, 38106 Braunschweig, Germany
e-mail: feedback@distributed-teams.com

large global project many feature teams work on the same product. Thus the product owners of the diverse feature teams need to collaborate so that the overall delivery is in the mind of the customer.

The third section discusses the role of the coach. This role ensures adherence and necessary changes to the agile process. In a large global project every individual feature team requires a coach and all the coaches together need to take care that the overall agility isn't lost in the interworking of all feature teams.

The next section looks into the various possibilities of ensuring conceptual integrity, or technical soundness of the overall system. Depending on the complexity of the project and the knowledge and skills of the project members, each feature team might require one architect. In other circumstances a single architect for the entire project might be enough. If more than one architect is required, the people taking this role have to work together. Otherwise conceptual integrity is out of reach.

The fifth section explains the responsibilities of a project manager in a large, global, and agile project. And finally the last section elaborates on the location of the various roles. Especially key roles should be close to the team they are supporting. The product owner steering a specific feature team should for example be co-located with the respective feature team.

19.2 Context

The experiences described in this chapter refer to several projects I have been working on. Moreover, I verified some of my experiences through frequent exchanges with colleagues of mine. Please find below some ranges for the characteristics of these projects:

- Most of my projects are rather large in terms of people, between thirty and 300 project members.
- The smallest distribution degree is two sites within one country and the biggest one six sites spread over the globe.
- The following countries were involved: Austria, China, Czech Republic, Germany, Hungary, Poland, Singapore, Switzerland, UK, USA.
- The domain for the system varied very much, we built systems for embedded products, financial applications, mechanical engineering, multimedia, and telecommunication.
- Although I refer to all of them as “projects” this doesn't cover the truth, because some of them are actually product development. The biggest difference between the two is that product development has no ending—as soon as the first version is shipped this product is maintained and additionally the team works on the next version of the product.
- Some of these “projects” were greenfield applications whereas other ones were existing ones with a lot of legacy.

I hope this context description helps in order to classify and better understand the experiences described and the recommendations given.

19.3 Configuration of a Feature Team

A single feature team should always be able to deliver whole business functionality (features or stories) or in other words business functionality should never be split across several teams.¹ Therefore, for a feature team to perform well the members of the team should either already comprehend, or be capable of acquiring the required knowledge that is necessary to complete a unit of business functionality. In addition to the domain and technical know-how that has to be present in each individual feature team, the team also requires the knowledge to actually deliver the functionality. In a typical feature team its members will fulfill the roles of architects, database administrators, designers, technical writers, domain experts, infrastructure specialists, integration experts, programmers, testers, and user interface designers.

Typically in agile teams, feature team members take turns in fulfilling these roles. There is hardly a single person taking the responsibility for only one role. Head monopolies—a few people who are regarded as the only experts for a specific area—have to be avoided, because they create a high risk. The project will have difficulties making any further progress if these people are for example on vacation, sick, or change jobs.

Ideally a feature team consists of seven, plus or minus two member (the Miller rule) and stays together for the whole lifetime of the project. Yet, in many projects the individual feature team size depends on the complexity and size of the domain area this feature team is responsible for.

The development of some distinct features might require a specific knowledge for only a certain amount of time. For example, in one of my projects we rarely required the expertise of database specialists for data migration (but most of the time this knowledge wasn't needed). We had therefore only a few migration specialists for the whole project, but not for every feature team. In such a situation, I recommend that this migration specialist gets part of the feature team only while the corresponding features are developed within this team. Very often such a feature team membership lasts only for one iteration.

Yet, preferably the required know-how should exist within each team, or else the team should be supported to build it up. Therefore, it is a good idea to use a similar approach if a feature team requires support for acquiring some specific know-how. In that case the respective mentor works with this team for as long as it is necessary to transfer the knowledge.

It often simplifies the job of these transient team members if they travel to the site of the feature team they're currently supporting. But, this is not always required especially not if the feature team is dispersed because such a team has by definition no common site. Therefore, for the support of a dispersed feature team—a team that is distributed in itself—the transient team members stay either virtually in contact with the people they are supporting or they travel to the sites where these people reside.

¹More on feature teams see Chapter 18: *Feature Teams—Distributed and Dispersed*.

19.4 Product Owner

Each feature team has to be made aware of feature prioritization and has to know whom to ask if there are problems in understanding the specifics of a feature. The agile manifesto requires in one of its principles that:

Business people and developers must work together daily throughout the project.

In other words: a feature team needs to collaborate with somebody representing the customer. To simplify matters, this representative is often labeled as *the customer*, or as in XP the *on-site-customer*. Most often this term is misleading, because this representative is seldom a customer himself, but somebody providing the business perspective in terms of the customer. For example, some systems need to serve various (competitive) customers who can't agree on the requirements. The Scrum term *product owner* is widely used for differentiating the real customer from the representative who collaborates closely with the feature team.

The product owner's task is to clarify the different requirements of the various customers and to decide on priorities. Thus, the product owner needs to know the business domain of the customer in detail and has to have a good communication channel to the (different) customers. The product owner is caught between two stools—he needs to support the feature team regarding the business knowledge and to involve the (real) customers to decide on priorities. This makes fulfilling the product owner's responsibilities very exhausting.

Most often product owners are recruited from different areas or departments, such as: marketing, support, product management, sales, or business analysis. If the end users of the system under development are developers, of course a developer is also an excellent candidate for the product owner.

19.4.1 Team of Product Owners

Most often there is a one-to-one mapping between product owner and feature team. The exception is if the system is rather simple and/or the feature team has built a similar system in the past, then a product owner can support several feature teams. For the norm, large global projects have a product owner for each and every feature team. These product owners need to synchronize the prioritization of the features so that the resulting system provides a surplus with every iteration. Ignoring the synchronization leads most likely to a system in which the contained features do not fit together or even contradict each other.

Thus, in a large global project the product owners have to work together as a (virtual) team. This team is claimed to be virtual, because the respective feature team is the "home" for every product owner. Moreover, most often the collaboration of the product owners happens virtually, because the product owners are situated at different sites.

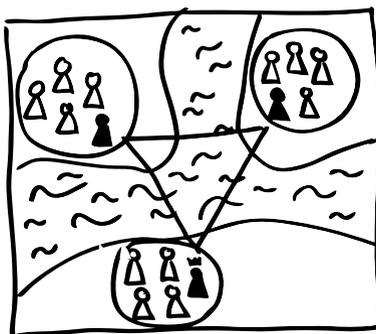
19.4.2 Lead Product Owner

Establishing a (virtual) team of product owners creates the risks that:

- Every product owner pushes his or her own features and ignores the overall business functionality of the system.
- Decisions on priorities are delayed, because the team can't come to an agreement.
- Different product owners contact the same customer with similar questions, which might upset that customer.

The core of these risks is that too many cooks spoil the broth. The more product owners belong to that team the more likely it is that the business perspectives differ. Therefore, the team of product owners has to be steered by a *lead product owner*, who mediates in case of discrepancy.

Fig. 19.1 Virtual team of product owners (in whole black) with lead product owner (symbolized with crown)



The lead product owner is the key contact to the (real) customers and collects their key ideas. The responsibility of the lead product owner is to spread these key ideas. Moreover, the lead product owner requires the input of the team of product owners in order to decide on business priorities.

Depending on the complexity of the system as well as of the project (based on size, degree of distribution and the like) the lead product owner might be able to additionally support one of the feature teams. That is next to the responsibility of leading the team of product owners, this person takes the role of an “ordinary” product owner. Yet, in most cases fulfilling the role of the lead product owner will be the unique (full-time) task of that person.

19.4.3 Collaborating with Both: Customers and Feature Team

The lead product owner is the key contact to the customers. Yet, for supporting their feature teams adequately—getting feedback and clarifying possible misunderstandings—also the product owners need to work together with the customers.

Preferably, every product owner is co-located with the feature team he's supporting. It's a rule of thumb that the more complex the business domain is the nearer the product owner has to be to the feature team. As a consequence, the product owner needs to work together with the customers virtually most of the time. At other times, he has to travel to the customers sites.

Obviously, if the feature team the product owner is supporting is not co-located but dispersed, he can't be co-located either but needs to travel to the various sites. Often it helps if the product owner is then at least situated at a site where a few members of the dispersed feature team reside. This way he will always be aware of the current situation. Whenever the product owner can't be close to his feature team the conversation between product owner and feature team has to be enriched by utilizing all kinds of communication media. The most important thing though is to try to reduce the times when the product owner is not close to "his" feature team.

However, it should be clear that there is no difference between an onshore and an offshore team. This is stressed, because sometimes it is assumed that offshore teams don't need a co-located product owner. A co-located product owner can ensure in the best way possible the growth of business value in the system. And delivery of business value is what agile is about.

If the assigned product owner can't be co-located with the feature team he is supporting—a different product owner is required. Global projects use often a shadowing concept to educate product owners at every site involved. This shadowing concept relies on experienced product owners at other sites who act as mentors for the inexperienced ones.

19.5 Coach—Also Known as Scrum-Master

The coach (Scrum term: scrum-master) ensures the agile process is supporting the feature team in the best way possible. If the process is hindering more than helping, the coach will work together with the team to improve it. Or whenever impediments occur that hinder the feature team on making progress, the coach will smooth this impediment out. An example would be escalating problems to the right people.

This doesn't necessarily mean that the coach is doing all this operatively himself, yet, he reminds the team that if something isn't working it needs to be changed for the better and asks every team member to responsibly do so. In this way, the coach acts more as a vivid reminder for the team. It should be the goal for every coach to become superfluous—and although I have never seen this happening in reality—this mindset helps the team to understand self-organization better.

In large global projects there is always a one-to-one mapping between coach and feature team. For every agile team there is only one exception to this rule: the team is perfectly self-organized and as a consequence the coach is superfluous. In all other circumstances every feature team is supported by one coach. If there is no experienced coach available, at least one of the team members has to grow into that role. The coaches of the different feature teams work together for ensuring the

overall agility of the project. They help feature teams to benefit from one another by transferring learnings, and good practices from one feature team to the others.

I find it important that the coach is actually a member of the team and not an outsider to the team for example by being one level higher up in the hierarchy than the team. To ensure this, the coach should support the team additionally as a developer or tester. However, the coach might not always be able to fulfill his responsibility as a regular team member—this depends on how well the team jells and how good it is in self-organization.

It should be obvious that the coach could fulfill his role best if he is co-located with the feature team he is supporting. Of course, for dispersed and thus not co-located feature teams the coach needs to communicate and collaborate with team members in different ways: by phone, e-mail and also by traveling. Moreover, similarly to the product owner, also the coach should be situated at one of the dispersed feature team's sites (and not a third site) in order to know what the team is struggling with.

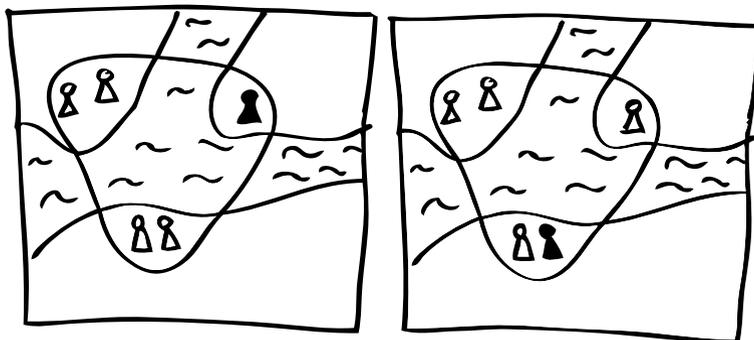


Fig. 19.2 Coach (in whole black) should be located at one of the dispersed team's site (right figure) and not at a third site (left figure)

19.6 Architect and Architecture

Small co-located agile teams typically take the responsibility for the architecture altogether. While concentrating on business features, close collaboration enables such a team to additionally ensure that the system is technically sound. This technical soundness is also called conceptual integrity and is defined by Fred Brooks as follows:

It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas. [1]

Conceptual integrity is according to Brooks the most important consideration in system design and the basis for simplicity and straightforwardness, which is in large global teams the main responsibility of an architect. For some projects it is sufficient for an experienced developer to additionally take the role of the architect. Yet in a large and global project you will need an experienced chief architect to ensure conceptual integrity across the entire project.

The need for architects depends on the technology, the technical know-how of the project members, the complexity and the size of the project. One architect advising in all technical decisions is enough for many global projects. This architect brings technical dependencies of features to the awareness of the business site which help the (lead) product owner to make the right prioritization decisions. In some cases the technical complexity of the system or the lack of technical know-how requires every feature team to get the support of one architect. Very often though, this support is only needed for the start of the project till the knowledge is built up. Then the responsibility of the architect morphs to regular development with only occasional pure architectural support. Finally, some projects benefit from a team of architects. In such a setting feature teams will get full-time support of an architect only for limited period of time, for example for the duration of one iteration before the architect moves on to support the next feature team.

Supporting a feature team doesn't mean that the architect provides concepts or documents yet it means he assists in implementing the features through actual coding.

19.6.1 Chief Architect

As soon as more than one architect supports the project it becomes crucial that the architects collaborate closely—otherwise conceptual integrity isn't guaranteed. Similarly to the virtual team of product owners, also the architects benefit from somebody—the chief architect—leading the group.

It is the chief architect who ensures that the big picture is communicated and understood well. He will act as the key contact for the business side and will keep the memory of key ideas alive [2]. Moreover, the chief architect spreads these key ideas and makes this way certain that more and more people gain the same understanding of the system. Without a chief architect feature teams (with the support of their architect) tend to suboptimize towards their own targets and lose sight of the total effect on the entire system [1]. An architect should never come up with concepts driven by self-fulfillment. Thus, it is the chief architect's task to convince all architects that they are providing a *service* for the feature teams that supports the development of business features.

Although the name of the role—chief architect—might imply that this person is dictating architectural decisions, this is far from being true. Yet, on the other hand using democracy like majority decision is also not a good advisor for making key decisions. Instead the guiding concept should be *nemawashi* defined by the Toyota Way as:

Make decisions slowly by consensus, thoroughly considering all options; implement rapidly. [3]

Taking this concept into account requires the chief architect to ensure that everyone gets heard, different views are evaluated and everybody is this way involved in decision making with the consequence that the final decision is backed.

19.7 Project Manager

For small co-located agile projects the classical tasks of a project manager are mainly performed by the product owner (a few are left for the coach). Yet, in a large and global setting the burden for the product owner(s) and coach(s) is already so high that these persons can't additionally take care of the organizational stuff. A critical responsibility belonging to that area is politics. A project can be in optimal shape but still be killed by ignoring political issues (or lobbying)—both externally and internally.

If a feature team requires specific resources, has difficulties accessing its product owner or customer, or if a project member wants to change sites or move to another department—the project manager can support effectively because his network inside and outside the company is typically more powerful (than for example that of a coach).

Often the responsibility for the budget lies also in the hands of the project manager. However, this requires close collaboration with the (lead) product owner.

Basically, it is the project manager who enables the entire project team by removing all impediments that can't be removed by the feature team members themselves.

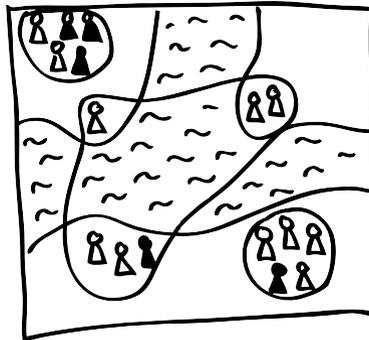
If the development of your global project is spread over several locations, it is unimportant at which location the project manager actually resides. He will have to travel to all the different locations anyway. However, if development is situated at one location and project management at a second one then that definition of project management clearly does not match the project manager role bed here. In such a situation I recommend project management to move to the development site for the duration of the project.

19.8 Key Roles Support Their Teams Directly

All key roles—especially coach, product owner and architect (if there is one per feature team)—should be co-located with their feature team. Actually, they belong to the team. Some organizations assign key roles to project members located at the site where the headquarters resides. However, this is by no means supportive. On the one hand, such a setting implies the key roles are more controlling than participatively serving the feature team and on the other hand, it feels like the headquarters mistrusts the other sites of being able to self-organize. Additionally, proximity makes collaboration more efficient.

Yet, if the respective feature team is dispersed there is no specific site where key roles should be located. The only rule to follow in such a setting is that key roles should reside at one of the dispersed feature team's site and not be solely located at yet another one. Still, the relation between efficient collaboration and proximity holds also true in a dispersed setting. Therefore, it is crucial for everyone playing a key role to travel frequently to all sites involved and to stay virtually in touch with the remote sites.

Fig. 19.3 Key roles (in whole black) should be co-located with their feature team no matter if the team is co-located or dispersed



19.9 Conclusions

Large global projects require special attention to roles and responsibilities. Each feature team should be structured in a way that all necessary knowledge and skills are either available within—or can be acquired by the team. Sometimes, for example during an educational period it might be necessary that a feature team requires the support from a mentor over a limited timeframe.

Every feature team is steered business-wise by a product owner. The product owner decides on the priorities of the features this team is working on and clarifies possible misunderstandings. In order to do so the product owner stays in contact to the (real) customer and to his peers. This team of product owners with the support of a lead product owner ensures that every iteration results in the delivery of a meaningful business value.

Every feature team is supported process-wise by the coach. Thus the coach ensures adherence to the agile process and if the process is inadequate he motivates the team to change the process for the better.

The focus on business value might lead into ignoring technical aspects. The architect brings these technical aspects to the awareness of the product owners. He explains technical dependencies between features which might influence their priorities. Depending on the project (in terms of size, content, technical complexity) every feature team might need the support of an architect or at the other extreme one architect might be sufficient for serving the whole project. If a project requires

more than one architect, the chief architect will ensure the synchronization of—and agreement on technical decision by all architects.

Contrariwise to small co-located agile teams, large global ones require a project manager making certain that the whole project runs smoothly organizational wise. The major concern of the project manager is to get the backing for the project both inside and outside the organization. This requires the project manager to deal a lot with politics.

Unless a feature team is dispersed, all key roles should be located at the same site as the feature team. In order to support a dispersed feature team the key roles need to be located at one of the team's sites, travel frequently to the various sites involved, and communicate virtually while away.

References

1. Brooks, F. P. Jr. (1995). *The mythical man-month: Essays on software engineering* (20th anniv. ed.). Reading: Addison-Wesley (Quoted from p. 42 and p. 44).
2. Cockburn, A. (2006). *Agile software development: the cooperative game* (2nd ed.). Reading: Addison-Wesley.
3. Liker, J. K. (2004). *The Toyota way. 14 management principles from the world's greatest manufacturer*. New York: McGraw-Hill (Quoted from p. 241).

Further Reading

4. <http://www.agilemanifesto.org>.
5. Eckstein, J. (2004). *Agile software development in the large*. Cambridge: Dorset House.
6. Eckstein, J. (2010). *Agile software development with distributed teams*. Cambridge: Dorset House.

Chapter 20

Getting Communication Right: The Difference Between Distributed Bliss or Miss

Jan-Erik Sandberg and Lars Arne Skaar

Abstract Communication is challenging in any IT project. In distributed projects distance, timezones and cultures are thrown into the mix making it even more challenging. By focusing on getting communication to work within these constraints as opposed to ignoring them, we have seen great results among those who have taken a pragmatic yet rigorous approach to making communication work—even in distributed projects. Although a significant additional cost of distributing the effort is still there—the cost can be managed and the disadvantage of distributing a project can be reduced by applying some best practices that are emerging.

20.1 Introduction

Of all that is difficult in standard software development projects, communication has always turned up as the most important and most challenging in our workshops on agile practices and while coaching agile teams.

Although communication is essentially difficult between teams, within the teams and towards external stakeholders, it gets even more difficult when adding cultural diversity, geographic distances and time zones into the mix. Consequently we have spent the last 5 years looking into how a team can get this right or at least improve from current practices. With the strong interest in making agile practices work even in distributed projects and the pragmatic approach we have seen from those team we have interacted with we strongly believe it is possible to counter these challenges.

J.-E. Sandberg (✉)
Det Norske Veritas, Bærum, Norway
e-mail: jan-erik.sandberg@dnv.com

L.A. Skaar
Miles, Oslo, Norway
e-mail: lars@miles.no

20.2 Background Overview

20.2.1 Background

Each of the authors behind this chapter have worked as agile coaches in multi-national software companies where off-shoring has been used extensively. Most of the recommendations are based on our experiences from these companies. In addition we have been running workshops at the international XP conferences (XP2005–XP2009) and Agile2008 in Toronto which have confirmed and augmented these experiences. Already in XP2005 and XP2006, we noticed that experience reports on distributed agile started to emerge. We decided to propose a workshop specifically on distributed agile at the XP2007 conference in Como, Italy based on that our workshop in XP2006 in reality gave most attention to this issue. With more than 20 participants this was obviously a relevant topic at that time. We would also like to acknowledge the contribution from Jutta Eckstein at that workshop who already had gained some experiences in dealing with this and having published a couple of books on the subject [1, 2]. The workshop has since then been run at XP2008 in Limerick, at Agile2008 in Toronto, Canada and at XP2009 in Sardinia, Italy.

We have summarized the characteristics of the companies we have worked with and some of their projects in the tables below. Not all companies and projects can be disclosed—still the domains they are in give guidance into the relevance of the experiences.

Table 20.1

Overview—Company 1

Det Norske Veritas, Norway	
Number of developers	100
When was agile introduced	2005
Domain	Classification of Vessels

Table 20.2

Overview—Company 2

Company: UKsoftware ^a	
Number of developers	100
When was agile introduced	2005
Domain	Banking

^aThe name is changed due to confidentiality reasons

Table 20.3 Overview—Project 1

Mortgage application

Duration:	1 year
Status:	finished
Agile practices:	Scrum, TDD, pair programming, continuous integration
Involved locations:	UK, Finland, India

Table 20.4
Overview—Company 3

Company: NORTelecom ^a	
Number of developers	50
When was agile introduced	2007
Domain	Telecommunication

^aThe name is changed due to confidentiality reasons

Table 20.5 Overview—Project 2

Self-service application	
Duration:	1 year
Status:	finished
Agile practices:	Scrum, TDD, continuous integration, frequent releases
Involved locations:	Norway, Czech Republic, India

20.3 Starting a Distributed Agile Project

Many organizations started their first offshoring efforts by relying heavily on formal written communication and formal hand-overs in the belief that this would overcome the communication challenges. For most of these organizations this proved not to be sufficient. Adding even more rigor and formality rarely helps—sometimes it might even make matters worse. Documents, email and contracts can only get you so far. You have to acknowledge the investment of effort that is needed to get collaboration to really work across distances.

Thus, many organizations are looking into agile practices to overcome the communication challenges; between team members, between teams and between stakeholders. Along with the need to adapt more effectively to requirements, to improve on quality and predictability through more frequent deliveries—which is a very transparent communication of what is being done. On quite a few occasions agile practices were strongly advocated by management on distributed projects due to failures with traditional approaches, such as waterfall. Due to the nature of distributed projects, the agile practices need to be adapted and augmented to counter the challenges of distributed teams. When agile practices were first introduced, it was assumed that co-location of development resources and even the customer was a strict prerequisite. However, this is a “luxury” that only rarely can be accommodated in a real-life scenario. In reality co-location is not an absolute requirement, but traditionally co-location has been used as a technique to improve communication in projects. Distributed projects needs to find other ways of coping. Simply ignoring it, does not really help.

Dividing work across locations is both a necessity and an opportunity to cope with this challenge. Jutta Eckstein covers popular ways of organizing work by teams and locations in her chapter. Her recommendation of organizing by customer-

requested business functionality is consistent with our findings. Consequently you need to establish team with the skill-mix necessary to do so in each location.

To be able to establish the correct and complete skill-mix at each location, an extra emphasis on skill transfer and building is likely to be necessary. Many organizations have experienced resistance from the onsite teams in facilitating such a skill transfer. It is likely that this is caused by fear of downsizing locally.

Furthermore you need to cope with cultural diversity. Although obvious, the importance is most often underestimated, and most organizations are usually surprised by miscommunication caused by not understanding the different cultures.

Practical Tip: Spend money on travel; be at the other location for an extended period of time; 2–3 months in order to really know the people. When spending money on travel, consider the following:

- Avoid spending all the travel budget on managers—although unfortunately that seems to be the normal priority.
- Prioritize travel to those who do the actual work in order to establish a good relationship with the other parties before working distributed.
- Travel from offshore location to onshore—the psychological effect of commitment to the organization and the project is usually stronger that way as you get a stronger sense of being part of something larger—rather than being a province.
- There is a risk of losing people after their “journey of a lifetime”—in that case consider a requirement to work for 2 years locally before being awarded travel.

Show that you care about people on the other side by celebrating birthdays, being respectful of local holidays, showing pictures of those on the other side—small things that matters significantly.

20.4 Low-cost and Effective Communication

From our experiences, it is crucial to think about getting the communication working, right from the beginning. All communication comes with a “cost” and it is a good idea to have a clear focus on how the project will provide for different types of communication. The cost associated with the technique should be measured against the value it provides.

As an example, discussion with your pair-programmer partner is something we consider as “medium cost”-communication giving high value. In a pair, you are already in the same thought domain and you are already sitting together. When you get to a point where you need to discuss something, everything is set up and all you need to do is speak. It is also important to notice that a lot of communication is

non-verbal (like body language and tone of voice). In a pair, all of this is included in the default communication pattern.

On the other side of cost, you have large meetings with dial-in members from multiple locations with different time zones. In this case, you have to schedule a meeting, often it is hard to find a time that works for everyone. You have to book a meeting room and make sure that the telecommunication solution works in all locations that will participate. Quite often there are troubles with the technical solution. More often than not (at least in the western culture) some people are late, causing non productive time for the other participants. When the meeting finally starts, it is often difficult to moderate the discussion in a way that gives high value to all participants. Very often people from the business side have a different degree of need for details, than programmers for example.

Be cautious of expensive video equipment installed in a meeting room. Due to the high investment costs associated with such equipment, there will rarely be enough for everybody. You would need to book in advance, you need to make sure the equipment work, you need to plan well ahead with those on the other side—and suddenly the cost of communication increases to a level where it is questionable how much help it really is. You are likely to have similar experience with high-end video conferences as with the dial-up conference calls, in addition to possibly even more formality and ceremony added to the use of such equipment. Most of those who have had access to such equipment reported less success than expected, although they did appreciate having it available. The figure below illustrates the differences in cost vs value added when comparing often used communication techniques in distributed projects.

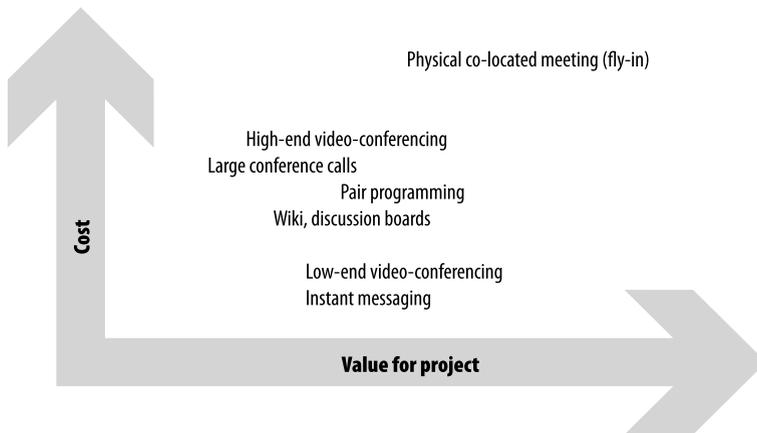


Fig. 20.1 Cost of communication vs. value added

Thus, high spending on communication equipment is not necessarily a guarantee for high value and success.

Practical Tip: Install always-on web cams and big screens at informal locations, such as by the coffee machine at each site. This way, it is quick and easy for the teams to discuss issues and they get the benefit of visual body language (if picture quality is good) and tone of voice. Most laptops come with web cams these days—those should be used. Buy one if they don't—they are cheap and easy to use. Software that supports video calls is widely available and you only need Internet access to communicate. Simple collaboration tools, like Wikis and discussion boards have also proved to be useful when working across distances.

20.5 Empower the Team

A while into the project, after all the fun of starting a new project, getting to know new people and working on a new solution, the project starts for real and so do the challenges to deal with. Although getting the build servers up and running, setting up the base architecture and getting to know each other is challenging in its own right, tougher challenges are likely to surface a few months later in the project. The distances between the teams starts to matter, and the separation of “us” and “them” starts to be important, not necessarily in a constructive manner, and not really intentionally either. This behavior often times leads to a decline in overall quality.

Practical Tip: Establish an on-going improvement process—across the distributed teams and within each team. We recommend weekly retrospectives both across teams and within teams regardless of the sprint length. In particular in the beginning of the project. This way we ensure that problems with for example quality and communication can be dealt with quickly.

At this point, we want to emphasize the importance of employing active leaders with a strong focus on keeping the team together. She need to continue the efforts of enabling good communication and tracking where the project is heading; even when the customer changes his mind. As with any projects, the “customer” typically realizes a while into the project that something else is needed or more than the funding allows is needed. This is challenging in a co-located project, making sure that the shift is properly communicated and implanted in a distributed project is even more challenging.

Practical Tip: Nurture common ownership and common responsibility for the success of the project across locations by celebrating successes together

and involving and empowering the teams in obstacles that arises during the project.

20.6 Common Architecture Across Locations

One of the principles behind the agile manifesto states “Continuous attention to technical excellence and good design enhances agility”. One of the implications of this principle is to ensure a common architecture regardless of the locations. Experience shows that there is usually a constant battle between keeping such a conformity against local initiatives to change or adapt the underlying architecture.

Practical Tip: Establish a central architecture function with a forward-leaning attitude, making sure that the common architecture always is attractive and useful for the development. Still avoid up-front architecture and opt for fall-out when needed. Tap into the skills of each location to leverage those into the common architecture as well as making sure it is suited for the problems at hand.

Communicate clearly whether an architectural decision is politically motivated or technically motivated. Doing so can reduce unnecessary tension and conflicts. For instance, choosing products from suppliers with a strategic relationship is usually more politically motivated than technically justified. It is usually better to be candid when this is the case as opposed to dressing it up as technically justified when it is not. Technically minded professionals most often will respect such decisions as long as the reasons behind are communicated clearly. Be honest and respectful towards the developers regardless of who they are or where they are located—to earn their respect and trust.

When setting up continuous build and integration across different sites, distance; hence latency makes a central common repository impractical in reality; regardless of how much bandwidth you have. The principle of having the continuous build running on the same code still holds.

Practical Tip: Set up replica of the central repositories that are replicated constantly to make continuous integration work in practice. This is also about communication—about communicating the code we are building to be tested and built upon as effective and often as possible.

Be aware of the high turnover in consultant companies providing offshore resources. The high turnover is considered among many of our subjects to be one of the most major drawbacks and risks when submitting work to offshore locations.

Extra sites and backup locations are recommended. Common practices and a common architecture across locations facilitate shifting locations when that turns out to be necessary.

Practical Tip: Extend the practice of common code ownership from eXtreme Programming to global code ownership to avoid depending on one particular team or location. Rotating responsibilities over assets, moving people around and transparency in general can contribute to this.

20.7 On “Proxies”

Using a proxy; i.e. an extra person or organization to access different aspects of the different locations is in practice unavoidable. Still, be conscious to the noise or filter being put in place when using the proxy, and avoid it when it is not needed. The risk of losing the effect of communication increases significantly when adding a proxy.

You are likely to have different kinds of proxies; management proxies and customer type of proxies. Many providers of offshore resources also promote a local supplier proxy to filter communication with the offshore resources. This has in most cases we have discussed turned out to be a bad practice; an anti-pattern which we strongly discourage. It leads to reduced empowerment of the offshored resources and too much information being lost in the communication.

Practical Tip: Avoid customer proxies if possible. If needed, consider empowering a senior developer or architect to act as a customer proxy and invest time in understanding the customer domain. Management proxies are a necessity; you need good strong local leaders to manage the completeness. You need good managers with a good understanding of the functionality to be delivered and being good at managing people; even if they are hard to get, this should be prioritized. Be conscious of the attitude you project to your developers through proxies—do you consider them code monkeys or creative individuals that make great software? You get the most out of your people if you treat them with respect in a humble way. Be aware that people from the Far East might have less respect for us in the west than we would think. Many of the resources from the Far East have been subject to a highly competitive culture all the way through their education system to their employment; thus it is usually the most skilled and educated people that we get access to. Thus, the proxy needs to be someone the local group respect; their leadership capabilities as well as their technical skills.

20.8 Conclusions

Through sharing experiences on how to cope with distributed projects we hope to see major improvements on such projects. By putting an emphasis on communication and applying agile practices to deal with the challenges associated with distributed projects, we are starting to see some best practices emerging. This chapters cover some essential areas to consider:

- Make sure the rational for running projects in a distributed fashion are understood and through that the rational for applying agile practices to deal with the challenges associated with distributed projects
- Focus on low-cost and effective communication—pragmatic use of widely available tools have turned out to be much more effective than high-end video-conferencing equipment; mostly due to the active use of them rather than burdening the use of such equipment with additional bureaucracy
- The importance of getting the sense of common ownership across locations and being respectful for those you collaborate with—invest in travel among developers and show that you care
- Invest in common architecture across locations
- Although you can't cope without proxies, be careful on how you use them.

References

1. Eckstein, J. (2004). *Agile software development in the large*. Cambridge: Dorset House.
2. Eckstein, J. (2010). *Agile software development with distributed teams*. Cambridge: Dorset House.

Chapter 21

A Task-Driven Approach on Agile Knowledge Transfer

Jörn Koch and Joachim Sauer

Abstract Constant and unimpeded communication is an essential ingredient when it comes to successful software development projects. While manageable if the team is within shouting distance, it poses a considerable challenge in global software development (GSD) projects. In this chapter we explore how a lightweight knowledge transfer process can be established between distributed development teams. The leit-motif of the transfer process is a hands-on approach that values actual cooperation on tasks over lecturing the learning team. It introduces a set of practices that take tasks as a central means to both drive the knowledge transfer and to integrate it with the ongoing development process. The practical relevance of the described practices was successfully experienced in a case study.

21.1 Introduction

Communication in distributed projects faces specific challenges. Organizational, geographical, and temporal gaps need to be bridged, which makes communication by far more difficult than in a co-located team (see [2, 4]). While impeded communication can be a roadblock for the distributed team to work cooperatively it is also very hard to involve stakeholders outside the team such as the customer in an agile fashion [3]. Though both aspects are important, we focus on overcoming the communication challenges within the distributed team exclusively and do not discuss how communication with remote customers is affected.

Good agile teams constantly adapt to changing requirements and conditions (e.g. influences from inside and outside the team). They have to share knowledge about the business domain, the customer's requirements, technical details of the implementation (most notably the software architecture) and established conventions and

J. Koch (✉) · J. Sauer
C1 WPS GmbH, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
e-mail: jk@c1-wps.de

J. Sauer
e-mail: js@c1-wps.de

practices. A significant part of this knowledge falls into the category of “tacit knowledge” that is hard to become aware of and, thus, can hardly be communicated explicitly.

In plan-driven projects, knowledge is commonly shared by bulky requirements documents that are hard to handle due to their sheer size. An agile knowledge transfer is instead characterized by the choice of lightweight practices and artifacts that fit in and support the agile progress rather than burdening it.

In this chapter we want to share what we have learnt about knowledge transfer in agile distributed projects. Our conclusions are based on a case study and our personal participation in distributed agile projects.

We focus on a task-driven process of knowledge transfer that was implemented in the case study and do not go into details of related agile practices and tools that were also used in the case study’s project such as distributed pair programming, shared wikis, usage of instant messengers, daily standup calls, etc. The task-driven practices that we discuss in the following have proven to be useful to share knowledge effectively in several projects within both distributed and co-located teams. The practices are easy to apply as they require little organizational adaptations and leverage existing agile techniques.

21.2 Case Overview

The goal of the case study’s project was to replace a web application for rail ticket sale that was developed by a third team over a period of three years. The new system was developed from scratch by an external German team of five people. It was successfully released after a development time of 300 person days. Due to strategic reasons, the project was scheduled to be handed over to a company-internal team in Poland for maintenance and further development. In this situation successful knowledge transfer was mandatory as it was a vital precondition to long-term project success.

Table 21.1 Company 1 overview

USATravel	
Number of developers	over 1,000 worldwide
When was agile introduced	2005 companywide
Domain	travel

Table 21.2 Company 2 overview

C1 WPS GmbH, Hamburg, Germany	
Number of developers	approx. 40
When was agile introduced	2000
Domain	software

Table 21.3 Project overview

Rail Ticket Sale	
Duration	24 months
Status	ongoing
Agile practices	XP, Scrum, TDD, Pair Programming
Involved locations	Germany, Poland, Ukraine

Table 21.4 Team overview

Locations	Number of members	Roles
Germany	3 (varying)	1 software architect/business analyst, 2 developers
Poland	3 (varying)	1 software architect, 2 developers
Ukraine	1	1 developer
Total:	7	

The actual transition was done in multiple stages: The first review of the new system by a Polish developer took place on-site right before the first release. It was followed by reviews in more detail, done by two Polish developers offshore. The remote Polish team then loosely observed the ongoing on-site development process for a period of approximately one year before the actual transfer started in April 2008, i.e. the Polish team joined the daily standup call without actively participating in the development or planning process. Most members of the Polish team were new to the project's business domain at that time. The transfer was concluded in December 2008.

Both teams consisted of both senior and junior developers in equal shares. Developers on all sites were well-versed in agile software development techniques (XP, AUP [1], and Scrum) and knew how to implement agile practices in projects. The senior developers had several years of practical experience in various agile projects in different roles. The agile methodology used in the project was XP supplemented and enhanced by scrum artifacts such as “product backlog”, “sprint backlog” etc., and TDD.

The teams could effectively communicate in English and/or German. Also, there were no noticeable cultural differences among the teams in passing and taking criticism which granted straightforward communication.

All participants considered themselves as one team that was stretched over multiple development sites, rather than separate national teams. The team sizes changed over time (see Fig. 21.1). The Polish team was supported by an additional developer from Ukraine. The slight time difference of one hour between Germany/Poland and Ukraine had no mentionable effect as it still allowed to align the working hours of the teams.

The actual size of the Polish team was larger than the number of persons that were directly involved in the transitioning process.

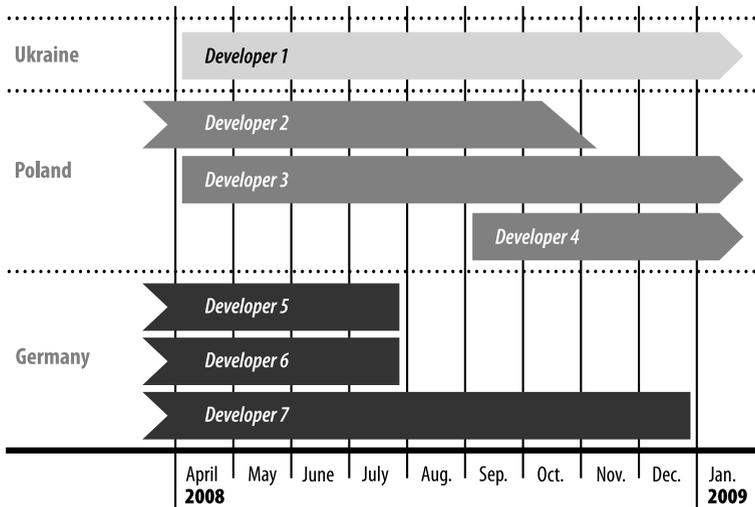


Fig. 21.1 The individual assignment of developers to the transitioning process over time

After the transition phase, the German team's participation was ended while the Polish/Ukrainian team successfully continued on their own.

The project had to tackle three major challenges: the complexity of the business logic, the hard-to-handle behavior of the rail supplier's external booking web service, and the rail supplier's tight release schedule. The system architecture was designed specifically to improve the comprehensibility of the business logic, to mitigate issues with the external web service, and to reduce maintenance costs caused by its mandatory updates. Maintenance was done along with further development, whereas maintenance meant adapting the software to those updates, as well as investigating production issues and applying bug fixes occasionally.

The system was taken into use in June 2007. Approximately 20 production updates were deployed until December 2008 including several minor and four major feature roll-outs. In December 2008 the system had reached an overall size of 80,000 lines of code including test code.

The system's architecture and code quality were considered "clean" as per the involved teams. Positive customer feedback indicated that the system was both suitable and mature from their perspective.

Debugging and bug fixing efforts were small and became negligible after the first 1.5 months of operation. In 2008 the longest bug-free period was 2.5 months. Critical bugs were never reported. The maturity of the system allowed a comprehensive transition of the project that was considered mandatory to ensure long-term project success. A less mature system had probably required focusing on its "dark areas" rather than on all parts of the project. From a practical standpoint, both scenarios require to apply the project's priorities to the knowledge transfer process.

21.3 Hands-On Approach (Task-Driven Approach)

The leitmotif of the transfer process was a hands-on approach that valued actual cooperation on tasks over lecturing the remote team. Architectural concepts and details about the process and the software were introduced as they came along the development path. This allowed focusing on topics that were obviously relevant in the project and also allowed supporting theoretical knowledge transfer by practical tasks.

The hands-on approach facilitated a very lightweight knowledge transfer process and encouraged its integration into the ongoing development process. It enabled the remote team to contribute to the development progress from the very beginning and saved the onshore team from preparing bulky presentations for didactical purposes.

This only worked because both teams contributed a fair amount of open-mindedness and mutual confidence. We consider this positive attitude mandatory for this approach. The offshore team was highly capable and willing to adopt conventions, rules, and practices that were applied in previous stages of the project and integrated those with established practices on their side. The onshore team was prepared to hand over responsibilities and to accept dissenting priorities of the remote team—in plain language: to let go of their former project.

In order to implement the hands-on approach, tasks became the central means as they define the actual units of activity in an agile process. We are aware of the fact that varying definitions of the task concept co-exist in practice. Any approach that defines tasks as small, self-contained action items that are required to implement a certain feature is compatible with our approach.

Practical Tip: Break down feature descriptions into tasks that

- contain all required information to execute the task,
- are goal-oriented (i.e. completing a task delivers a pre-defined result), and
- are small enough to be completed by a pair of developers in a couple of days.

As feature descriptions may be vague at an early stage do not try to define a complete set of tasks upfront. Instead add tasks whenever they emerge during the incremental implementation and shaping process of a feature.

We established four task-based practices: joint task planning, question-driven task scheduling, adequate task design, and scrupulous task sign-off.

In the following paragraphs we will describe those practices and their contribution to agile knowledge transfer.

21.3.1 Joint Task Planning

Tasks were planned jointly by both teams. Technically this was done on conference calls in combination with desktop sharing in order to collaboratively review, schedule, and assign tasks in XPlanner and to review documents and source code if necessary. In addition to the usual joint iteration planning that was also done on a regular basis tasks from the shared sprint backlog were often scheduled and assigned ad hoc during distributed pair programming sessions (see end of 21.3.3).

Joint task planning proved to be an ideal technique to implement the hands-on approach as it touches on all aspects of the project. Starting from this central technique promotes a common understanding of the project, mitigates the risk of misunderstandings, decreases churn and makes developers identify with their tasks.

Our experience confirms the findings that voluminous and detailed documents such as comprehensive feature descriptions do not help to reduce the required amount of communication. Instead, it turned out to be more important to enable developers to make the right decisions when working on tasks on their own rather than specifying tasks as exactly and comprehensively as possible.

The planning process itself led to a common understanding of the project rather than the resulting plan did. Also, this process was the starting point the following techniques were derived from.

21.3.2 Question-Driven Task Scheduling

Along with the joint task planning, work was rather shared than distributed between both teams. Whichever team was capable of undertaking a due task was a potential candidate for assignment.

Maintenance tasks usually were of high priority in combination with a deadline, and they also required a very profound understanding of the software. Thus, at the beginning of the transitioning process only the on-site team was capable of handling maintenance tasks while the remote team focused on less critical tasks instead. The more experienced the remote team became the more they started working on maintenance and other critical tasks.

Even though the differing project knowledge of the teams limited the possibilities of task assignment, a notion of a “leading” and a “following” team was not considered helpful. Instead both teams steered the knowledge transfer process by their means: the on-site team by their project knowledge, the remote team by their questions. Thus, in addition to the business priorities that applied as usual the mandatory knowledge transfer added its own priorities to task planning.

The remote team was very clear about which parts of the project they needed to understand better. It became obvious that asking questions proactively was a basic prerequisite to incorporating knowledge transfer priorities into task planning.

Practical Tip: In case a learning team hesitates to ask questions, it needs to be encouraged! It is essential for the learning team to be aware of their responsibility to acquire all the required knowledge they depend on in order to do their job.

Be aware of the fact, that the learning team may simply be overwhelmed by the complexity of the project not knowing where to start asking questions. Focus on small self-contained tasks then and leave everything else out of scope.

Cultural or individual hindrances to ask questions may be hard to detect and to mitigate. It can help to formalize the way of asking questions and establish a “project culture” based on a defined set of practices the team agreed upon.

Within the confines of the overall business priorities the remote teams’ questions had a direct impact on scheduling and assigning tasks, i.e. the priority of a task would be pushed up if this helped to dwell on a topic in order to deepen the understanding. This allowed the remote team to acquire self-contained chunks of knowledge rather than jumping erratically from topic to topic.

Over time the remote team had fewer and less pressing questions. The absence of pressing questions on one side that can actually be answered by the other side is clearly a strong indicator that the knowledge transfer is done.¹

21.3.3 Adequate Task Design

Along with pushing up the priorities of interesting tasks we found that it is best to provide remote developers with as much information as they can chew at a given time and let them arrive at their own insights when working on the code. We found in other projects that over- or undersized tasks often turn into both motivation killers and poor results (either through intimidation or overconfidence) while tasks in-between challenge and inspire most, leading to the maximum amount of forward-leading questions.

Adequate and coherent tasks enabled developers to deliver good results that have the potential to become their personal sample solutions for similar tasks in the future.

¹It may seem hard to tell the difference between the knowledge transfer being complete, and the learning team just being reluctant to ask questions. As the learning team is required to effectively work on tasks from the very beginning you can usually tell by their results. If the results are good and the learning team does not ask questions there may have been no need for a knowledge transfer in the first place. If the results are poor and yet the team is not asking questions, it is a strong clue that the knowledge transfer is not yet complete. Note that if a team delivering no more than poor results is permanently reluctant to ask questions the knowledge transfer is done anyway and will probably have failed.

Tasks that were coherent in reference to their business requirements made them both better understandable and easier to communicate. Also, their sign-off was apparently very meaningful as they had coherent acceptance criteria assigned. Thus, it was a constant challenge not to end up with incoherent tasks when tailoring tasks of adequate complexity. E.g. in case a complex task involved user interface and internal programming it turned out to be a better idea to break it down into smaller chunks of business requirements rather than introducing separate tasks for user interface and internal programming.

In case an adequate task design could not be found, e.g. because a task introduced complex new topics, we stuck to the original task design and implemented it in distributed pair/group programming sessions using remote desktop sharing while communicating via phone. This worked out very well after a certain adaptation phase and was done more often than the task design actually required. We noticed that in smaller projects with less than about five iterations tailoring tasks for knowledge transfer is often hard and distributed pair-programming is an appropriate alternate practice.

21.3.4 Scrupulous Task Sign-Off

The learning effect was intensified by a scrupulous sign-off process that took into account all aspects of software quality related to business requirements, architecture requirements, code style, test coverage, etc.

It is understood that the sign-off gives obligatory feedback on the implemented solution. It also turned out to be a very valuable and effective technique to communicate conventions, rules, and practices that were unfit to be communicated explicitly as they may have been applied unconsciously or were simply diffuse. E.g. printouts were rather straightforward to implement from a technical perspective, however, the criteria for a good layout never needed to be discussed nor had there been a need before to document those. After the offshore team implemented their first printout solution the result was signed-off in an iterative process that both improved the layout step-by-step and clarified which layout design criteria actually applied (e.g. certain parts of the layout should match existing printouts, certain information should not be shown in certain tables to reduce visual complexity, etc.).

The scrupulous sign-off process provoked enlightening discussions and helped both teams to become aware of how the project actually worked. In fact, it helped to emerge diffuse and implicit conventions as explicit rules.

Whenever necessary, those rules and conventions were documented by the remote team in a so-called end to end documentation along with technical topics.

21.4 Conclusion

The task-driven practices of the hands-on approach were rather discovered than invented. We observed their practical relevance in our case study and found that

those practices integrated well with the ongoing development process and promoted an effective and lightweight knowledge transfer process. At the end of the project transition phase, the Polish/Ukrainian team successfully continued on their own and from today's perspective proved that they will do so in the future.

Joint task planning is the starting point of the described task-driven approach. It is a prerequisite to incorporating the questions of the learning team into task planning. Those questions lead us to the most interesting, thus, most relevant tasks in regard to the intended knowledge transfer. However, we found that tasks can be unfit to be assigned to a learning team either because they do not have a suitable size or do not cover coherent parts of the business requirements. Thus, we pointed out that tasks need to be tailored to be adequate for knowledge transfer.

At the end of the task-driven approach the scrupulous sign-off intensifies the learning effect as it both gives valuable feedback on the implemented solution and also helps to communicate "tacit knowledge".

Task-driven agile knowledge transfer relies very much on effective communication (planning tasks jointly, posing questions, discussing tasks and their sign-off, distributed pair-programming, etc.). Language barriers are a severe hindrance in global software development projects that can probably make our approach inapplicable.

Differing cultural styles of communication (especially the handling of criticism) can be a similar challenge as severe interpersonal misunderstandings are a common implication that can lead affected team members to practically avoid communication. Our approach can probably still be applied here, if more formalized and boiled down to a set of roles and practices. The idea is to establish a shared "artificial" culture of how to interact in a task-driven agile knowledge transfer process.

Acknowledgements The authors would like to thank all participants of the discussed project for their support in the preparation of this chapter. Special thanks go to Lukasz Pielak and Andreas Kornstädt for their valuable suggestions in improving the text.

References

1. Ambler, S. W. (2006). The agile unified process (AUP), <http://www.ambysoft.com/unifiedprocess/agileUP.html>.
2. Christiansen, H. M. (2007). Meeting the challenge of communication in offshore software development. In *Proceedings of the 1st international conference on software engineering approaches for offshore and outsourced development (SEAFOOD 2007)* (pp. 19–26). Revised Papers, Zurich, Switzerland, February 5–6.
3. Korkala, M., Pikkarainen, M., & Conboy, K. (2009). Distributed agile development: A case study of customer communication challenges. In *Proceedings of the 10th international conference on agile processes in software engineering and extreme programming (XP 2009)* (pp. 161–167). Pula (Sardinien), Italy, May 25–29.
4. Kornstädt, A., & Sauer, J. (2007). Tackling offshore communication challenges with agile architecture-centric development. In *Proceedings of the 6th working IEEE/IFIP conference on software architecture (WICSA'07)*. Mumbai, India, January 6–9.

Chapter 22

Architecture-Centric Development in Globally Distributed Projects

Joachim Sauer

Abstract In this chapter architecture-centric development is proposed as a means to strengthen the cohesion of distributed teams and to tackle challenges due to geographical and temporal distances and the clash of different cultures. A shared software architecture serves as blueprint for all activities in the development process and ties them together. Architecture-centric development thus provides a plan for task allocation, facilitates the cooperation of globally distributed developers, and enables continuous integration reaching across distributed teams. Advice is also provided for software architects who work with distributed teams in an agile manner.

22.1 Introduction

Global software development projects deal with a number of challenges, including communication and coordination between the teams, requirements and knowledge transfer, and division of responsibilities [5]. Agile methods offer approaches to many of these challenges as is discussed in other chapters of this book. A substantial area of agile distributed projects is that of coordination between the teams. While agility offers general advice and some supporting practices, many projects could benefit from additional detailed procedures for working together while being distributed. In this chapter I propose architecture-centric development as a means to strengthen the cohesion of distributed teams by providing a common artifact and related practices, thus further enhancing the positive effects of agility. It will be shown that architecture-centric development may also facilitate cooperation between teams that operate at unequal levels of agility. The chapter includes sections on the related agile practices of continuous integration and collective ownership as well.

J. Sauer (✉)
C1 WPS GmbH, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
e-mail: js@c1-wps.de

22.2 Case Overview

The conclusions in this chapter are based on my research at the university of Hamburg and personal experience with distributed agile projects and various interviews with project managers, architects, and developers who have taken part in various different agile distributed projects. In this chapter I describe a case study in detail, incorporating results from other studies where appropriate.

The case study dealt with the development of an innovative application in the mail order business. While the project's scope was rather restricted both in duration and in number of participants, it nevertheless was an important trial balloon for agility and offshoring and the customer expected valuable insights for future development projects.

The development work was outsourced to a German onshore team and an Indian offshore team. The onshore team consisted of three to five developers and one architect who also did the requirements engineering and was the overall project lead. Up to six members were part of the offshore team. The development took four months of distributed work following an agile process mainly built on XP (see [10] for details).

Table 22.1 Project overview

Project A	
Duration:	4 months
Status:	finished
Agile practices:	XP, TDD
Involved locations:	Germany, India

Table 22.2 Team overview—Germany

Location	Number of members	Roles
Germany	4–6	1 software architect and project lead, 3–5 developers

Table 22.3 Team overview—India

Location	Number of members	Roles
India	5–6	1 team lead, 4–5 developers/testers

The project started with a joint kick-off meeting at the customer's headquarters and a co-located first iteration onshore. Two Indian developers took part in this iteration. They had only limited prior experience in agile development. All developers

were trained in the customer's domain, the applied agile process and practices, and in the necessary technology. The Indian developers returned to their home country after the first iteration and established the development team there, passing on their knowledge.

Some weeks into the development process, problems with the agile process emerged, as the Indian team was not able to work as agile as planned, in some parts falling back to familiar plan-driven practices. This led to misunderstandings with the breakdown of tasks and ultimately to a decrease of the application's quality. The offshore team took long to accomplish relatively simple tasks. The integration of code into the common version control system became tedious and error-prone.

The onshore project lead tried to exert influence on the offshore team and to assist with unit testing and other agile practices, but these measures did not enhance the situation. It became apparent that another supporting approach was needed. Therefore, architecture-centric development was enforced to strengthen the cooperation between the teams and to improve the application's overall quality. The members of the onshore team had gained positive experience with this practice in co-located projects, so it seemed appropriate to adopt it.

22.3 Software Architecture and Architecture-Centric Development

22.3.1 *Software Architecture*

Before exploring the facets of architecture-centric development, we should define our understanding of software architecture and its meanings in practice. Bass et al. define software architecture as “the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationship among them” [2]. Every system has a software architecture but only an active occupation with it exploits advantages.

Smolander identified four metaphors to describe the general meanings of architecture in practice [14]. These metaphors should not be seen as alternative views but as four parts of the same concept:

Architecture as blueprint: The software architecture is seen as a blueprint for all development activities. It serves as a universal plan for the implementation, deployment and further development of a software system.

Architecture as literature: The architecture description contains technical knowledge about the structure and design foundations of a system. It serves as the main document for understanding a system and facilitates its maintenance and further development.

Architecture as language: Software architecture establishes a common language for communicating about a system and its development. This enables different stakeholders to discuss requirements of a system and its implementation together.

Architecture as decision: A system's architecture emerges as the result of design decisions that are often made under the influence of conflicting goals and trade-offs. The architecture serves as reference of these decisions and constitutes the basis for future decisions.

22.3.2 Architecture-Centric Development in General

With architecture-centric development, the software architecture takes center stage in the development process. Most other tasks are aligned along the software architecture. The roots of architecture-centric development can be found in the Unified Process [8] and in work at Carnegie Mellon's Software Engineering Institute [1].

Architecture-centric development brings many advantages (cf. Smolander's metaphors): It helps in gaining better insight into the development process and the state of the project. The architecture description serves as a common object of work that all participants use and understand. By referring to terms and concepts of the architecture, all stakeholders use a common language that is closely related to the development work. This facilitates precise discussions and arrangements and reduces misunderstandings.

Examination of a system's architecture assists in breaking down the development work into smaller tasks that can be carried out as independently from other tasks as possible. Dependencies between different teams and thereby the need for communication and coordination decreases. Architecture-centric development also helps in achieving a balance between rigid requirements and individual responsibilities and skills of the development teams.

22.3.3 Architecture-Centric Development in Agile Distributed Settings

Software architecture in distributed projects is treated in the literature mainly from two points of view: To split the work into meaningful tasks that can be implemented by distributed teams [7] and as common artifact that allows the teams to coordinate their work [11].

From the analysis of the incidents in the case study and other examined projects as well as my personal experience I can conclude that the benefits of architecture-centric development can be utilized to an even greater degree in distributed projects that follow an agile philosophy. Beyond the software architects' elemental task of designing and maintaining the software architecture, I can identify three comprehensive functions which are especially useful in agile distributed development:

- Software architects combine domain knowledge with technological knowledge. It is crucial in distributed projects that knowledge is maintained in the project and transferred in an easily comprehensible manner to all team members.

- Software architects are able to find a balance between rules and individual responsibilities of the developers. It may be hard to develop common rules for procedures and programming germane to all teams implicitly. A software architect with overview of the system and his technical knowledge can establish those rules.
- Software architects can moderate between and counsel teams. In nearly all of the projects that I analyzed, quite a lot of issues arose between the teams, many of technological nature. As direct communication is hampered in distributed settings, software architects can act as mediators.

Agile projects generally do not depend on extensive documentation but on close, flexible cooperation of all stakeholders. It may be argued that architecture-centric development adds more formalism and more documentation to an agile project as the software architecture is yet another artifact that has to be maintained. On the other hand, architecture-centric development significantly eases coordination between the distributed teams and enables better insight into the state of the development. The description of the software architecture is a document that has to be maintained anyway and does not add too much overhead in consideration of the advantages.

To fit with a flexible, agile approach, care should be taken that the architecture is not developed entirely up-front and does not restrict the development work more than necessary. Therefore, the structures below the high-level architecture should be evolved from the base through the developers without consulting the architect beforehand. The changes should later be checked by the architect who can incorporate them into the official architecture documentation. Automated tools can support this otherwise tedious and error-prone task [3].

22.4 Distributed Continuous Integration and Collective Ownership

Architecture-centric development facilitates the agile practice of continuous integration. The fact that all teams manage to contribute their work to a sound combined system may be the only indicator that a project is still on track. This explains the important role of the integration progress in distributed development. With several distributed teams integrating their code into the same common version control system, a shared architectural vision helps to ensure integrity and improves the software's inner quality.

Continuous integration entails many advantages, mainly with regard to risk reduction [6]: Integrations happen frequently with relatively small amounts of new code. Long, error-prone integrations are avoided. Every developer has instant access to the latest code enhancements. Bugs are generally noticed quicker and are therefore easier to fix. As an up-to-date version of the developed system is always available, releases may be deployed frequently. These risks are aggravated by not working co-located, so distributed agile teams benefit from continuous integration to an even greater extent.

Practical Tip: Some agile methods advocate a separate “integration machine” in one place that provides a clean environment for integrations. This is plainly not possible in distributed environments. In practice, this hardware machine can be substituted without greater loss by software integration guards like Cruise Control that are accessible by all teams.

A little effort has to be put into the selection and setup of a common version control system, especially when some teams in less developed countries are forced to work without a reliable communication infrastructure. In these cases, buffering version control systems that are able to bridge network dropouts should be used. An interesting question is whether continuous integrations scales to larger projects. I could mainly gain experiences with this practice in small to medium-size distributed projects during my research. But other authors also report positive results in large-scale projects (see [12]). A related agile practice is collective ownership. This practice allows every developer to change any resource, most notably the source code. No one owns pieces of code, everyone is equally responsible for the integrity and quality of the codebase. In my research it became apparent that many distributed teams have problems with this practice, mainly because trusting your own work to external, hardly known colleagues presents psychological barriers for many developers. As a consequence, individual or team-level code ownership was often reintroduced for relevant parts of the code. A look at open-source communities may help to understand practical options and alternatives of collective ownership.

22.5 Practical Advice for Software Architects

Because of the architecture’s importance in global software development, I recommend to explicitly define the role of a full-time software architect in distributed projects. It is important to involve the software architect during the whole lifespan of the project, right from the start. The architect should establish a reasonable usage of architecture early in the project and regularly check and adapt the procedures, if necessary. This way it is easier to gain acceptance from all teams and to avoid negligence that could gradually lead to problems with the software’s inner quality.

Practical Tip: In large-scale projects, each development team should be guided by a software architect to enable timely decisions and to avoid bottle necks. A chief architect should define and review the high-level architecture and adjust general design and architecture principles to be employed. He coordinates and moderates the work of the distributed architects. Also see Chap. 19 by Jutta Eckstein on this subject.

From the analysis of different projects with distributed agile work, I can derive five problem areas that software architects in distributed projects should keep an

eye on. These areas encompass vital and challenging duties and responsibilities of architects working with distributed teams:

Architecture evolution and documentation: The development of an architecture should be seen as an iterative process, not as up-front task that establishes unalterable structures. The further development of the architecture has to be guided by architects to ensure that it stays suitable for the system. Future requirements have to be taken into account. The architecture has to be documented in a way that it is understandable by all participants [4]. The up-to-date architecture documents have to be accessible from all sites, e.g. by putting them into a common version control system or wiki.

Inner quality and unit tests: The system's inner quality has to be checked on a regular basis because internal problems and quality trade-offs of the distributed teams might take some time to surface otherwise. Many of the analyzed projects showed problems with insufficient unit tests so particular attention should be directed to this area.

Convey the requirements: Business and technological requirements have to be discussed with the developers. In distributed settings with reduced communication bandwidth, user stories may not be sufficient. If this is the case, stories should be augmented with more context and other information like acceptance tests etc.

Supervision and guidance of the development: It can be hard to gain insight into the progress status of other teams. The teams should have sufficient space to arrange their own development processes. Then again the overall development process should stay manageable and controllable.

Assignment of tasks: The tasks should be divided between teams in a manner that they can work independently from each other. By splitting the tasks along appropriate architectural units, the dependencies between the teams can be reduced.

Based on my studies, I can also give some advice for the design of software architectures that are well suited for agile distributed projects.

Practical Tip: As a general rule, one should stick to proven design and architecture principles: information hiding, loose coupling, strong cohesion, design by contract, open closed principle, avoidance of type interdependencies, etc. These principles gear towards understandable software with fewer dependencies, thus easing maintainability and further development. They have proven to be especially valuable in distributed projects where developers have limited possibilities to communicate with each other and the main source of information is the source code.

When choosing a concrete architecture style, emphasis should be placed on simplicity. The architecture should support remotely working developers rather than presenting a source of misunderstandings and complexity.

Practical Tip: Components, services, and layers are well-established building blocks of sound architectures. Good architects are proficient in breaking a system down into these building blocks while at the same time advancing the high-level architecture. A well documented model architecture that is built on these principles and building blocks can help in this task. Especially, if it is backed up by a sophisticated framework [9].

22.6 Conclusions

In this chapter I argued how architecture-centric development may help in agile distributed projects. An alignment of processes along the lines of the software architecture enables the teams to tackle general challenges of distributed projects, e.g. breakdown and distribution of tasks, a common language for communication across sites and project-wide rules and standards. I could observe that distributed projects profit from an architecture-centric approach.

The usage of architecture-centric development allowed the teams in the case study's project to work together in a structured and consistent way while internally relying on divergent processes and practices: the onshore team utilized an exemplary agile process while the offshore team reverted to a more plan-driven process. The onshore team used pair programming, test-driven development, daily stand-up meetings, and other important agile practices while the offshore team used single programming and divided programming and testing tasks between different developers.

Exchanging team members between the sites (aka dual-shoring), further helped to establish a sound overall process (see [13]). These measures, together with expanded quality assurance procedures, helped to set the development back on the right track. Feedback from both teams showed a general acceptance of the readjusted development process. Delays and problems in one team affected the other team to a lesser degree than before. Ultimately, the project could be brought to a successful end on time and on budget. While the overhead of integrating an offshore team without prior record of working together was rather high in this rather small project, valuable experience could be gained. Thus, foundations were laid for follow-up agile projects with globally distributed development teams.

References

1. Bass, L., & Kazman, R. (1999). *Architecture-based development* (CMU/SEI-99-TR-007). Carnegie Mellon University.
2. Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Reading: Addison-Wesley Longman Publishing.

3. Bischofberger, W. R., Kühl, J., & Löffler, S. (2004). Sotograph—a pragmatic approach to source code architecture conformance checking. In F. Oquendo, B. Warboys, & R. Morrison (Eds.), *Proceedings of the 1st European workshop on software architecture (EWSA 2004)* (pp. 1–9). St Andrews, England, May 21–22.
4. Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002). *Documenting software architectures: Views and beyond*. Upper Saddle River: Pearson Education.
5. Damian, D., & Moitra, D. (2006). Global software development: How far have we come? *IEEE Software*, 23(5), 17–19.
6. Fowler, M. (2006). Continuous integration. <http://www.martinfowler.com/articles/continuousIntegration.html>.
7. Herbsleb, J. D., & Grinter, R. E. (1999). Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5), 63–70.
8. Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process*. Reading: Addison-Wesley Longman Publishing.
9. Kornstädt, A., & Sauer, J. (2007). Mastering dual-shore development—the tools & materials approach adapted to agile offshoring. In B. Meyer & M. Joseph (Eds.), *Proceedings of the 1st international conference on software engineering approaches for offshore and outsourced development (SEAFOD 2007)* (pp. 83–95). Revised Papers, Zurich, Switzerland, February 5–6.
10. Kornstädt, A., & Sauer, J. (2007). Tackling offshore communication challenges with agile architecture-centric development. In *Proceedings of the 6th working IEEE/IFIP conference on software architecture (WICSA'07)*. Mumbai, India, January 6–9.
11. Ovaska, P., Rossi, M., & Marttiin, P. (2003). Architecture as a coordination tool in multi-site software development. *Software Process: Improvement and Practice*, 8(4), 233–247.
12. Rogers, R. O. (2004). Scaling continuous integration. In *Proceedings of the 5th international conference on extreme programming and agile processes in software engineering (XP 2004)* (pp. 68–76). Garmisch-Partenkirchen, Germany, June 6–10.
13. Sauer, J. (2008). Enabling agile offshoring with the dual-shore model. In W. Maalej & B. Bruegge (Eds.), *Software engineering 2008—Workshopband, Fachtagung des GI-Fachbereichs Softwaretechnik* (pp. 35–42). Munich, Germany, February 18–22.
14. Smolander, K. (2002). Four metaphors of architecture in software organizations: Finding out the meaning of architecture in practice. In *Proceedings of the 2002 international symposium on empirical software engineering (ISESE 2002)*. Nara, Japan, October 3–4.

Part V
Epilogue

Chapter 23

Agility Across Time and Space: Summing up and Planning for the Future

Darja Šmite, Nils Brede Moe, and Pär J. Ågerfalk

Abstract In this epilogue chapter the authors revisit the book content and identify the emerging trends in understanding the application of agility across time and space. This book concludes with the findings from an expert survey that put summarize the most important practical advice and the major areas of improvement and future work.

23.1 The Beginning of the End

This book has been devoted to the concept of agility across time and space. More specifically it has addressed the motivations, challenges and strategies used when organizations are adopting agile methods and practices for distributed projects. In this final chapter we aim to summarize the state of the art and point out some important future work. We do so based on two sources. First, the chapters in this book clearly represent up-to-date knowledge in the area that also reflects what people are currently working on. The chapters were initially submitted in response to an open call, then subjected to a thorough peer-review process, and finally selected for inclusion based on their overall contribution and quality as deemed by the reviewers (including both other authors and external experts) and the editors. To complement these, we also conducted a small Delphi-inspired study as part of the production

D. Šmite (✉)
Blekinge Institute of Technology, Ronneby, Sweden
e-mail: darja.smite@bth.se

N.B. Moe
SINTEF ICT, Trondheim, Norway
e-mail: nilsm@sintef.no

P.J. Ågerfalk
Uppsala University, Uppsala, Sweden
e-mail: par.agerfalk@im.uu.se

process. At the end of the book project we invited all accepted authors to suggest the three to five most important advice to give to an organization that is transitioning to agile distributed development, and the three to five most important areas for improvement/research in relation to agile distributed development. Altogether 12 authors replied with a total of 47 advice and 39 areas. We compiled these, eliminated redundancies and generated common phraseology. This resulted in 21 advice and 10 areas for improvement/research. The authors were then invited to rank these in order of importance and also to suggest up to three additional items that they thought were missing from the current lists (three advice and three areas). In addition to the ranked lists, we received in total 14 additional advice and 9 additional areas for improvement/research. This material, along with the chapters, was then qualitatively analyzed¹ using Atlas.ti, and the results were as follows.

23.2 Current Themes

When looking at the chapters of this book a number of central themes emerge. First of all, it is clear that most chapters deal with different aspects of organizing agile distributed development: project management, architecture, planning, and different aspects of teamwork come to the fore. This is certainly not surprising given the overall theme of the book. Emphasis is put on the management of inter-team dependencies, which can be reduced by paying attention to the relationship between software architecture and team structure. Two other themes that are central throughout the book are changing requirements and the soft, people-related, factors. Indeed, these themes are at the core of agile methods in general, and apparently play an important role also when moving into agile and distributed modes of working. A lesson to learn is that turbulent business environments and changing requirements emphasize the importance of understanding customer needs in terms of business value. In order to achieve such understanding, evaluation of the economic value of user stories and improved financial models are needed. Among the people-related factors, communication stands out as central to the success of distributed agile development. Communication is of course central to understanding customer needs, but also to facilitating teamwork and engagement. Finally, the chapters present various strategies to handle process- and practice related challenges, such as pair story authoring and careful consideration of subcontracting practices.

23.3 Practical Advice

In terms of practical advice, three high-level messages stood out clearly from our Delphi-inspired study. First, make sure to develop a sense of teameness by valuing people-factors and put measures into place to integrate teams across locations.

¹Although we followed customary qualitative analysis practices (recursive abstraction; open and axial coding), we do not claim the results to be anything else than indicative and food for thought.

Providing appropriate tools and technology can facilitate this. Enabling face-to-face collaboration when needed, valuing cultural differences instead of seeing them as a threat, enabling teams to deliver complete functionality independent of localization, and acquiring skilled human resources at each location and be prepared to reconfigure rapidly are other helpful advice.

Second, make sure to tailor your development processes to the agile and distributed context and to explain to everyone involved why things are done the way they are. Make sure to treat the transition into agile distributed as a change process in its own right, communicate the benefits and forecast possible problems. A good idea is to start small with a selection of pilot projects. When tailoring the process a number of practices are worth considering to grow both culture and mutual respect within the team:

1. Employ effective introspection and retrospectives with defined metrics that can be used to promote the increased improvement in productivity of the development team.
2. Find ways to minimize temporal delays, for example by adopting appropriate tools for daily cross-site scrum meetings.
3. Use short iterations and do continuous integration on a common deliverable across locations.
4. Apply intensive mandatory team-wide code inspections of earlier iterations.

Third, it is advised to ground management practices in established management theory and to educate management in order to ensure proper management support. This is important in order to secure appropriate resources and to establish effective reporting and management structures across sites. In doing this it is important to make sure that financial responsibility matches an agile approach. In the spirit of agile, it is furthermore important secure and maintain customer involvement using well-defined acceptance criteria.

Table 23.1 Summary of the most important advice

Practical Advice

Meet face-to-face, co-locate for a while, exchange members, and avoid distribution if possible.

You need patience and stamina.

Provide appropriate tools and communication infrastructure to support rich and intense interpersonal communication.

Concentrate on how everyone is helped by agile distributed and give careful consideration to dispersion.

Make people enthusiastic and don't be religious on the method; adopt and adjust what is justified and explain why.

Enable teams to deliver whole functionality (establish teams independent of the location).

Seek true customer involvement, overcome unavailability of and maintain continuous connectivity with the customer.

Ensure ability of the team to collaborate with each other without significant temporal delays.

To sum up, the table above presents the top eight advice as resulted from the responding authors' ranking. These eight were the suggestions that received a median ranking less than 10, which indicates relatively high consensus that these are the most important things to think about when transitioning to agile distributed development.

As can be seen from table, a final word of caution is, be prepared to work hard to get it right—you will need patience and stamina!

23.4 Areas for Improvement and Future Research

The areas suggested by the responding authors primarily centered around four themes: process, tools, business and research approach. With regards to process, it was highlighted that we need better to understand how to tailor agile methods for the distributed context. One way to start would be to try to understand the contingencies surrounding agile distributed projects; how do, for example, project size, business domain, and different team configurations affect the application of agile. Is there a difference between greenfield development and expansions of legacy systems in terms of the configuration of the development process? How can we improve communication across temporal, geographical and socio-cultural distance?

Tools need to be developed to support development practices, teamwork, and management of agile distributed projects. While there are many existing tools out there, they need to be developed in light of a better understanding of the conditions under which agile teams work in distributed projects.

In terms of business, two primary areas were identified. First, there is a great need to improve financial models in order to support the goals of the agile organization rather than just project management metrics, or ordinary business metrics. Second, these models need to be aligned with developments in agile contracting. This is a problematic area in agile development in general, and is certainly not less problematic in a global business context, which may involve several legislations and norm systems related to doing business.

The lack of theoretical models of agile and distributed development needs to be addressed. It is certainly true for agile in general that theory is lagging behind practice, and the distributed context is no exception. Developing theoretical models probably requires studies from both a micro perspective (addressing team dynamics) and from an organizational macro perspective. It was furthermore suggested to study the combinations of team members from different geographical, cultural, and temporal areas that appear to employ agile development effectively versus those who experience greater difficulty using this approach.

Although this book has arguably contributed significantly to our understanding of agile distributed development, much more is needed in order to learn how to apply agile methods to distributed projects effectively. Most importantly, we need to understand better why certain practices work or does not work, and in what particular contexts. The only suggested area for research/improvement that stood out in the responding authors' ranking as more important than the rest was: *Understand*

the effects of agile distributed development. While the chapters in this book provide a lot of useful advice for how to cope with agile distributed development, the actual effects of this mode of software development remains to be seen as the number of projects, successful and unsuccessful, increase.

23.5 The End of The End

The aim of this concluding chapter was to summarize the state of the art in the area of agile distributed development and to point out some important areas for future work. While a lot has been achieved, much more needs to be done. We started working on this book project because we believed that there were many lessons to be learned from current research and practice in the area; lessons that had not yet been collected into a text that combined actionable advice for the practitioner with captivating insights and challenges for the academic. We believe that we have achieved that with this book. We also hope that the content of the book will serve as food for thought that can trigger new developments and exciting opportunities for further development of the area.

Index

40 hr work week, 120

A

Agile Distributed Development

- á la carte approach, 6
- benefits, 40, 62
- effect, 146
- issues, 5, 136, 142, 204, 211, 225
- pitfalls, 77
- state of the art, 6

Agility, 107

Agility dimensions, 113

Analytic hierarchy process, 109

Architect, 296

Architecture, 209, 217, 226, 229, 286, 295, 307, 321, 323

Architecture-centric development, 231, 324

C

Case study, 11, 31, 47, 91, 117, 133, 149, 167, 171, 201, 217, 235, 259, 301, 311, 321

CMM, 33, 93, 101

Coaching, 127, 294

Coding standards, 19, 37, 209

Collective ownership, 35, 37, 120, 209, 326

Composition, 221, 230

Concentration, 153

Conceptual integrity, 283, 285, 296

Continuous flow, 35

Continuous integration, 24, 37, 145, 245, 250, 302, 325

Coordination, 221, 321

Customer, 201, 281, 293

- expectations, 119, 123, 146

involvement, 26, 202

tests, 35, 37

D

Data dictionary, 103

Data model, 103

Distributed pair/group programming, 318

E

Economic value, 60

F

Feature list, 103

Feature team, 282, 291

- dispersed feature team, 283, 291, 294, 295

- distributed feature team, 282

Feature-driven development, 93

Fixed-price, 98

Formal processes, 101

H

Heterogeneous process environment, 126

High level planning, 172

I

India, 322

Integration, 221, 230

Integration-oriented approach, 218

Interruptions, 153

Italy, 154

Iteration planning, 37, 155, 177

Iteration reviews, 35, 37

K

Knowledge transfer, 312

L

Large-scale project, 11, 48, 134, 220

Limited work in progress, 37

Linear development, 280

Location

Asia, 224

Australia, 110

Austria, 280, 290

Baltics, 73

Brazil, 35, 37

Canada, 37

China, 37, 168, 280, 290

Czech Republic, 261, 280, 290, 303

Denmark, 173

East Europe, 73

Europe, 48, 110, 135, 224

Finland, 236, 261, 302

France, 168

Germany, 135, 261, 280, 290, 313, 322

Greece, 135

Hungary, 280, 290

India, 93, 110, 117, 135, 174, 206, 302

Ireland, 206

Japan, 110

Latvia, 261

Malaysia, 261

Mexico, 37

Norway, 73, 261, 303

Poland, 110, 280, 290, 313

Russia, 37, 261

Singapore, 110, 280, 290

South Africa, 110

Switzerland, 110, 280, 290

the Netherlands, 110

the United States, 11, 35, 37

the United States, 93

the United States, 110, 135, 168, 206, 224, 280, 290

Ukraine, 313

United Kingdom, 37, 110, 280, 290, 302

M

Mentoring, 122

N

Nemawashi, 296

P

Pair programming, 302, 313

Pairing, 37

Personnel selection, 120

Plan-driven, 177

Planning game, 16, 59, 120, 125

Planning poker, 39

Pomodoro technique, 149

Process-centric coordination, 224, 226

Product owner, 286, 292, 293, 296

Progress tracking, 244

Project planning, 149, 244

Promiscuous pair story authoring, 47, 55

R

Refactoring, 25, 37, 209

Retrospectives, 35, 37, 271

Round-the-clock, 103, 168

S

Scrum, 35, 37, 93, 135, 174, 209, 242, 259, 302, 313

daily meetings, 35, 37, 73, 97, 120, 125, 209, 260

product backlog, 95, 242, 272, 313

product owner, 144, 242

scrum-master, 294

sprint, 209, 242, 266

sprint demos, 270

sprint planning, 125, 209, 268

sprint reviews, 209

sprints, 95

Scrum-of-Scrums, 264

Self-organization, 294

Self-organizing teams, 37

Shared ownership, 245

Short iterations, 73

Simple design, 21, 209

Small releases, 27, 37, 303

Story points, 58

Storyboards, 125

Subcontractors, 239

Subteams, 281, 282, 286

Sustainable pace, 22, 35, 37, 152

T

Task planning, 316

Task scheduling, 316

Tayloristic approach, 163
Teaching, 122
Team structure, 280
Technical service team, 285
Test-driven development, 37, 302, 313,
322
The United Kingdom, 154
Time estimates, 58, 149, 152
Time management, 151, 167
Time-boxing, 152
Timeshifting, 168
Tools, 210, 276
Traditional development, 206, 217
Training, 120

U

Unit testing, 24, 327
United, 93
User stories, 327

V

Version control, 245

W

Waterfall, 33, 280
Whole team, 35, 37

X

XP, 12, 23, 37, 120, 135, 154, 292, 313, 322