The Purposeful Adaptation of Practice:
An Empirical Study of Distributed Software Development

Anders Sigfridsson

Supervisor: Prof. Liam Bannon

Thesis submitted to the University of Limerick for the degree of Ph.D.

Department of Computer Science and Information Systems, University of Limerick, May 2010.
Abstract

The Purposeful Adaptation of Practice: An Empirical Study of Distributed Software Development

This thesis addresses the research question: how do software developers in distributed projects continually develop their practical knowledge through dealing with changing circumstances in daily work practice? The purpose is to complement previous research on knowledge-related issues in distributed software development settings by assuming a practice-based perspective on knowledge and studying knowing as seen in software developers actual everyday activities.

This work builds on extensive field studies of a team in a multinational organisation and of an open source community called PyPy. The primary methods utilized were participant observation and grounded theory coding. The thesis provides thick descriptions of the two cases, coupled with vignettes that illustrate how issues of evolving technology, enculturation and socialisation of new members, and emerging project purposes and goals manifested in the settings under study.

What is being argued is that the software developers in the studies dealt with changing circumstances through the purposeful adaptation of practice. Evidence is presented that shows how this is an ongoing process of practices adopted to allow for change and change stimulating adjustment of practice. When the software developers do this, there are at least two important concerns: anticipating changing circumstances and alignment of activities.

This empirical study thus unveils underlying dynamics of knowledge development that matter for how sharing actually happens in these settings. The temporal dimension of knowledge is emphasised and attention is shifted towards supporting software developers contemplative activities. The findings also opens up for an alternative understanding of what situated means in distributed software development, showing that, as well as locale-specific practices, it may also be understood in terms of professional context and shared enterprise. Further light is also shed on the complexities of distributed software development practice. Distribution is shown as a mundane issue, as something that is dealt with as a collaborative challenge alongside many other issues. With the focus on changing circumstances, a previously unexplored aspect of work in these settings is characterised.
Declaration

The Purposeful Adaptation of Practice: An Empirical Study of Distributed Software Development

By Anders Sigfridsson

Supervisor: Prof. Liam Bannon

This thesis is presented as fulfilment of the requirements for the degree of Doctor of Philosophy at the University of Limerick, Department of Computer Science and Information Systems, Faculty of Science & Engineering.

It is entirely my own work and has not been submitted to any other university or higher education institution, or for any other academic award in this University. Where use has been made of the work of other people, it has been fully acknowledged and referenced.

Signature

Anders Sigfridsson
Acknowledgements

There are a lot of people who have to be mentioned for their importance to me. In no way exhaustively, nor in any indicative order, here they are:

My supervisor Prof. Liam Bannon, whose great academic heritage has come to pervade my work and what I have written in this thesis.

My examiners, Prof. Yvonne Dittrich and Dr. Xiaofeng Wang, who thoroughly and fairly assessed my work and provided truly constructive feedback during the viva voce.

My father, mother, sister, brother, relatives and friends back home. Knowing that you're all still up there no matter what and that I can come home any time has been the most valuable thing I've learnt during my years abroad!

My colleagues and friends from the socGSD project, who have been a great support and inspiration. Special thanks to Anne Sheehan, who I have worked closely with and whose help has been a truly invaluable contribution; and Gabriela Avram, who has been very influential in my thinking and always pushed me on beyond my self-imposed limits!

All the people in, and associated with, the Interaction Design Centre. Cheers for the laughs, the slagging, and the good times! Special thanks to Luigina Ciolfi and Cristiano Storni, who both kindly have read and eloquently dismantled everything I imposed upon them; Erik Lövquist and Nora O'Murchu, with whom I have had countless fruitful discussions; Anne Murphy, who really made it all happen; Michael Cooke, who has taught me most of what I know about women and (unrelated) how to handle bears (in theory); and Marc McLoughlin and Colm McGettrick, who have been a great support.

All the people in the various institutions that have funded and kindly harboured me during my time as a PhD student, particularly LERO (The Irish Software Engineering Research Institute) and the Department of Computer Science & Information Systems at the University of Limerick. Special thanks also to the people in the Department of Informatics at the University of Umeå, who nudged me onto the PhD path armed with a solid education.

The people at SEGAL research facility in the Computer Science Department at University of Victoria, British Columbia, and those at Umeå Institute of Design, both with whom I have had a valuable exchange of experience.

All the people who I met during my field studies, for putting up with me and contributing so greatly to the insights embedded in this thesis.

Thank you all!
# Table of Contents

Abstract............................................................................................................................i

Declaration.......................................................................................................................ii

Acknowledgements...........................................................................................................iii

List of Tables...................................................................................................................viii

List of Figures...................................................................................................................ix

1 Introduction..................................................................................................................1

   1.1 Objectives and contributions................................................................................1
   1.2 Summary of research............................................................................................2
   1.3 The phenomenon of distributed software development..............................4
   1.4 The concept of practice.......................................................................................6
   1.5 Thesis outline......................................................................................................7

2 Issues of knowledge and change in distributed software development........10

   2.1 Introduction............................................................................................................10
   2.2 Knowledge-related issues in distributed software development..............11
      2.2.1 Knowledge in software development...........................................................11
      2.2.2 Knowledge sharing and mutual knowledge.............................................12
      2.2.3 Situated and embedded knowledge.............................................................15
      2.2.4 Complex social and collaborative processes..........................................16
      2.2.5 Summary.......................................................................................................17
   2.3 A practice-based perspective on knowledge......................................................17
      2.3.1 The problem of commodification.................................................................17
      2.3.2 An epistemology of practice and “practical knowledge”............................18
      2.3.3 Practical knowledge as inescapably flexible and evolving........................20
      2.3.4 Summary.......................................................................................................22
   2.4 The need for investigations of how knowledge is developed through practice...22
      2.4.1 The need for in-depth studies of knowledge-related issues in practice......22
      2.4.2 The development of practical knowledge through dealing changing circumstances..............................................................................................................23
      2.4.3 Summary.......................................................................................................24
   2.5 Topics for studying changing circumstances in distributed software development practice........................................................................................................24
      2.5.1 Improvisation, emergence and change – a perspective on practice........25
      2.5.2 Prominent topics of change in distributed software development........27
         2.5.2.1 Evolving technology..........................................................27
         2.5.2.2 Enculturation and socialisation..............................................29
         2.5.2.3 Emerging project purposes and goals......................................31
      2.5.3 Summary.......................................................................................................33
3 A qualitative investigation of software developers work – methodology and research methods

3.1 Introduction

3.2 Research context and methodological background

3.2.1 The socGSD project and group

3.2.2 Focus and methodology of the socGSD project

3.2.3 Summary

3.3 Research perspective

3.3.1 Qualitative perspectives on software developers

3.3.2 The potentials of qualitative approaches

3.3.3 Summary

3.4 Empirical and analytical methods

3.4.1 The research challenge: unveiling work practice

3.4.2 Participant observation – the groundwork

3.4.3 Grounded theory coding – coping with the flood

3.4.4 Field studies and thesis layout

3.4.5 Summary

3.5 Chapter summary and conclusions

4 Elucidating distributed software development practice – distribution as a mundane issue

4.1 Introduction

4.2 The Lambda field study – the workaday in a distributed project

4.2.1 The Lambda project

4.2.2 Research method – an observational field study

4.2.3 Concerns of a software development team in a distributed project

4.2.3.1 Solving problems together

4.2.3.2 Coping with remote dependencies

4.2.3.3 Adjusting practices

4.2.3.4 Going off track – with a purpose!

4.2.3.5 Dealing with uncertainties and change

4.2.4 Summary and discussion

4.3 The PyPy case study – interweaving online activities and collocated events

4.3.1 The PyPy open source project

4.3.2 Research process – an exploratory case study

4.3.2.1 Embarking on an exploration

4.3.2.2 Gaining an insiders view

4.3.2.3 Looking for the constitutively specific

4.3.2.4 Ascertaining the participants perceptions
4.3.3 The role of sprints in PyPy.................................................................80
  4.3.3.1 Synchronous work and project rhythm........................................80
  4.3.3.2 Facilitating stronger social relationships.....................................82
  4.3.3.3 Attracting and enculturating new members...............................83
  4.3.3.4 Deferring complex or controversial discussions........................85
  4.3.4 Summary and discussion.............................................................87
4.4 A comment on the comparison of organisational and open source software development activities.................................................................89
4.5 Summary of chapter findings............................................................90

5 Exploring issues of change in technology, participants, and goals................91
  5.1 Introduction.....................................................................................91
  5.2 Selection and representativeness of the vignettes............................92
  5.3 Vignette 1: Lambda development team dealing with technological contingencies ..........................................................93
    5.3.1 Lambda development team deciding whether to update or not........94
    5.3.2 Lambda development team dealing with the framework change......95
    5.3.3 Lambda team members reflecting on the incident.......................96
    5.3.4 Summary and discussion.........................................................97
  5.4 Vignette 2: The introduction of a novel collaboration platform in the Lambda project..........................................................98
    5.4.1 Introducing Accord to the Lambda project..................................98
    5.4.2 Contingencies prompting caution among Lambda team leaders.....100
    5.4.3 Lambda team evolving shared practices for the new platform.......101
    5.4.4 Co-adapting Lambda team and Accord system roles..................103
    5.4.5 Contingencies staggering the work flow of the Lambda team.......105
    5.4.6 Summary and discussion.........................................................106
  5.5 Vignette 3: Participation and enculturation in the PyPy open source community ..........................................................108
    5.5.1 A newcomer joining the community.........................................109
    5.5.2 The newcomer attending his first sprint.................................110
    5.5.3 The newcomer meeting the gurus – and their expectations?.........114
    5.5.4 Summary and discussion.........................................................115
  5.6 Vignette 4: The PyPy community negotiating project goals and external expectations..........................................................116
    5.6.1 PyPy deciding what development areas to prioritize ................116
    5.6.2 PyPy’s initial response to external expectations........................119
    5.6.3 PyPy deferring the discussion and outsiders perceiving it as “answer by silence”.................................................................120
    5.6.4 Summary and discussion.........................................................121
  5.7 Summary of chapter findings...........................................................122

6 Characterising how change in circumstances unfolds..............................125
  6.1 Introduction.....................................................................................125
6.2 Continuous change ........................................................................................................ 126
  6.2.1 Discussion ............................................................................................................. 128
6.3 Equivocal situations .................................................................................................... 129
  6.3.1 Discussion ............................................................................................................. 131
6.4 External influence ....................................................................................................... 132
  6.4.1 Discussion ............................................................................................................. 133
6.5 Chapter summary and conclusions ............................................................................ 134

7 Framing the purposeful adaptation of practice as knowledge development ............. 137
  7.1 Introduction ............................................................................................................. 137
  7.2 The purposeful adaptation of practice .................................................................... 138
    7.2.1 An ongoing process ......................................................................................... 138
    7.2.2 Two important concerns ............................................................................... 141
      7.2.2.1 Anticipation .............................................................................................. 141
      7.2.2.2 Alignment ............................................................................................... 143
    7.2.3 Discussion ........................................................................................................ 145
  7.3 Purposeful adaptation from a practice-based perspective on knowledge ............. 147
    7.3.1 Rephrasing adaptation of practice as knowledge development ................. 147
    7.3.2 Discussion ........................................................................................................ 149
  7.4 Chapter summary and conclusions ........................................................................ 151

8 Conclusions and reflections .......................................................................................... 153
  8.1 Summary of thesis structure ................................................................................. 153
  8.2 Thesis conclusions ................................................................................................. 154
  8.3 Contributions .......................................................................................................... 155
    8.3.1 A view on distribution as a mundane issue .................................................... 155
    8.3.2 Characterising changing circumstances ......................................................... 156
    8.3.3 Extending research on situated knowledge ..................................................... 156
  8.4 Areas for further research ...................................................................................... 157
  8.5 Limitations and lessons learned ............................................................................. 158
  8.6 Personal reflections ................................................................................................. 161
    8.6.1 On the concept of knowledge and information systems design .................... 161
    8.6.2 What is “Global Software Development”? ..................................................... 163

Bibliography ..................................................................................................................... 168

Appendix A: Distributed software development – research landscape and key issues .......................................................................................................................... 183
## List of Tables

Table 1: The potentials of qualitative approaches.......................................................... 41
Table 2: Summary of findings from Lambda field study................................................. 68
Table 3: The phases of the PyPy case study................................................................. 73
Table 4: Summary of findings from PyPy case study..................................................... 88
Table 5: Summary of findings from the four vignettes.................................................. 123
List of Figures

Figure 1: An overview of the structure of the thesis.........................................................9
Figure 2: Sites and relationships of the Lambda project...............................................56
Figure 3: Image of codes during second iteration of coding in Lambda case..............58
Figure 4: Images from PyPy sprint in Limerick, Ireland, August 2006......................74
Figure 5: Images from PyPy sprint in Hildesheim, Germany, March 2007.................75
Figure 6: Sequence of events in vignette 1.................................................................93
Figure 7: Sequence of events in vignette 2................................................................99
Figure 8: Sequence of events in vignette 3.................................................................108
Figure 9: Sequence of events in vignette 4.................................................................117
Figure 10: Three aspects of the changing circumstances..........................................135
Figure 11: The purposeful adaptation of practice.......................................................145
1 Introduction

1.1 Objectives and contributions
In this thesis, I focus on work practice in distributed software development projects and the issue of how software developers continually develop their practical knowledge while accomplishing their work. I assume a practice-based perspective on knowledge (Cook and Brown 1999; Orlikowski 2002), emphasising that what practitioners know is constantly being updated as they engage with work tasks in flexible and continuously evolving contexts. To explore this, I examine in detail how software developer groups in distributed settings deal with changing circumstances. Thus, the research question of my thesis is:

- How do software developers in distributed projects continually develop their practical knowledge through dealing with changing circumstances in daily work practice?

Building on empirical material from extensive field studies, work practice in two different settings – a multinational organisation and an open source community - is elucidated for the purpose of addressing this question. The term “practical knowledge” is here used in reference to what the software developers know about how software development can be done within particular contexts. With “circumstances” I mean the conditions under which they perform their work, primarily in terms of the constraints, opportunities, and resources which they can dispose in their activities.

Knowledge-related issues are identified as significant in the literature on distributed software development (Priklanicki, Audy et al. 2003; Desouza, Awazu et al. 2006) and they are the subject of much empirical research (e.g. Cramton 2001; Imsland and Sahay 2005). In this discourse, there is a growing emphasis on the importance of situated aspects of knowledge (Sole and Edmondson 2002; Nicholson and Sahay 2004). This
means that knowledge is intricately tied to the particular circumstances where it is applied. However, whilst issues related to the sharing of situated knowledge have been investigated in recent literature, there is still a scarcity of studies on how software developers' knowledge is maintained in particular contexts within distributed projects. This is an empirical shortcoming, as previous research on related issues rely primarily on retrospective accounts of practice. I attend to this gap by *unearthing the dynamics of knowledge development in these settings* through qualitative observational studies of software developers' actual activities.

In doing so, I shed further light on the complexities of practice in these settings, particularly showing distribution as a mundane issue and how it is dealt with as a collaborative challenge alongside many other issues. With my focus on changing circumstances, I highlight and characterise a particular aspect of work that has not been previously explored in these settings. This provides an increased understanding of distributed software development projects as constantly changing environments, where available technology and resources are in a constant flux, and goals, relationships, and practices continuously evolving. But most importantly, in unveiling this aspect and interpreting it as knowledge development, I extend the research on knowledge in distributed software development by uncovering the contingencies that matter for how software developers' competence is developed through practice. This refines our understanding of how knowledge sharing actually happens in these settings.

### 1.2 Summary of research

My research has taken place within the socGSD project,¹ which focused on social, organisational, and cultural aspects of distributed software development. The research I did represents one individual strand of the socGSD project, but it was undertaken very much within the frame of the project and with the support of the research group. Within this context, two field studies have been conducted for this thesis, utilizing participant observation and grounded theory coding as core methods. The first case is a development team of a globally distributed project in a multinational organisation, where I performed participant observation across a period of 5 months. The second is an

---

¹ “Social, Organizational, and Cultural aspects of Global Software Development”, a joint project between LERO (Irish Software Engineering Research Institute) and the Interaction Design Centre at the University of Limerick, Ireland. socGSD was one of the LERO cluster projects funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI).
open source community called PyPy, which was a major longitudinal case study, extending over 2 years in time.\(^2\)

By juxtaposing a review of research on knowledge-related issues in distributed software development and literature on a practice-based perspective on knowledge, I argue that a fruitful area for further research is knowledge as seen in what the software developers actually do. In particular, I am contending that a significant contribution can be made by studying how they deal with changing circumstances. With a lens on practice that emphasise emergence and improvisation (Orlikowski 1996; Weick 1998; Ciborra 1999) and an extended literature review on the complexities of software development as an activity, I suggest three topics as relevant for exploring this:

1. Evolving technology - because of its pervasive role in the phenomenon under study as both the object and tool of work (Sahay, Nicholson et al. 2003; Carmel and Tjia 2005) and how technological evolution is intricately tied to changes in organisation and practice (Orlikowski 1992; Orlikowski 1993).

2. Enculturation and socialisation of new members - because of the transient nature of software development projects (Brooks 1995; Sim and Holt 1998; Krogh, Spaeth et al. 2003; Ducheneaut 2005), and since a practice-based perspective on knowledge gives priority to matters of participation (Brown and Duguid 1991).

3. Emerging project purposes and goals - as both academic writings on software development in practice (Lanzara 1999; Luna-Reyes, Zhang et al. 2005) and the contemporary movement towards agile methodologies among practitioners (Fowler and Highsmith 2001; Dybå and Dingsøyr 2008) highlight the constant flux of project objectives.

My approach for exploring these topics is to provide thick descriptions of the software developers daily activities based on empirical material produced through field studies. I first elucidate work practice with analytical categories that emerged through grounded theory coding. Then I extract four episodes of practice – two from each case - and

\(^2\) The thesis does not concern itself with evaluating the differences between distributed development in global organisations and that in free open source communities. Instead, the choice of cases relate to the objective of the thesis, which is to characterise the development of practical knowledge in distributed software development practice. I argue that there are fundamental similarities in the challenges facing software developers in these two different settings, and that the differences in setting give rise to contrasts that are useful in my analysis. This is a recurring point throughout my thesis, but see especially section 4.4 for an empirically based discussion on the analogy of my cases.
render them as vignettes that illustrate how issues of change in technology, participants, and goals manifested in the particular settings under study. By showing what shared themes emerged across the vignettes, I formulate a characterisation of how changing circumstances unfold in distributed software development practice. Following on from this, I turn attention to how the software developer groups in my studies dealt with this, assembling evidence unearthed throughout my empirical analyses and interpreting that in terms of practical knowledge being developed in response to changing circumstances. What I show is that the situations facing the software developers is characterised by continuous – as compared to discrete and distinct – change, associated with external influence – particularly technological and professional relationships – and resulting in equivocal situations which the groups have to act within. The argument I make is that they deal with this through what I call the purposeful adaptation of practice, an ongoing process of practices adopted to allow for changing circumstances and changing circumstances stimulating adjustment of practice. I show that the purposeful adaptation of practice is one of the processes through which the software developers develop their competence in response to changing circumstances.

1.3 The phenomenon of distributed software development

I will take a step back now and provide some background information on the phenomenon that I study and my perspective on it.

In contemporary society, there is an increasing prevalence of relationships and exchange that transcend distance, a trend related to the rapid evolution of information and communication technologies, but also to social, economic, and organisational developments (Webster 2006). Distributed work – that is, work involving direct (either real time or asynchronous) communication and interaction between geographically distributed participants – has become a prominent element in many different domains (Hinds and Kiesler 2002). One domain where such trends are manifest is software development, where there has been a widespread shift towards distribution of collaborative efforts.

In corporate settings, global software organisations, outsourcing relationships, and inter-organisational collaboration are commonplace (Sahay, Nicholson et al. 2003; Carmel and Tjia 2005). But one must also consider open source software development an
important part of this trend, as widely dispersed and heterogeneous communities of developers have produced very complex and widely used software products such as the Mozilla Internet suite, the Apache web server, and the Linux operating system (Mockus, Fielding et al. 2002; Fitzgerald 2004). The distinction between corporate and open source (volunteer-based) settings is becoming fuzzier. Open source concepts and values are to an increasing extent inspiring changes of mainstream development models, while open source communities themselves are becoming economically viable and technically competent options for carrying out large-scale development projects (Fitzgerald 2006).

Distributed work is a topic that can be understood on many different levels, from individual levels, to group and organisational levels, to industrial and societal levels (Hinds and Kiesler 2002). Correspondingly, distributed software development is a topic within many different academic discourses, particularly in computer science disciplines such as software engineering, information systems, and Computer-Supported Collaborative Work, but also in areas of organisation studies, management studies, and economics (see Appendix A for more details on the research landscape). My perspective on the phenomenon is that of the information systems community (Avgerou 2000), viewing it as a matter of distributed groups performing systems development in real life contexts. The overarching purpose of my work is to provide detailed empirical analyses of practice which may be used to support distributed software development efforts by informing information systems design approaches.

Large-scale software development projects are characterised by high complexity, both in terms of the systems under development, and of the collaborative effort itself. For the latter, particularly issues of communication and coordination have long been given attention by scholars (Curtis, Krasner et al. 1988; Kraut and Streeter 1995) and practitioners (Brooks 1995) alike. It is now also argued that increased geographic distribution exacerbates such issues in a number of characteristic ways (Grinter, Herbsleb et al. 1999; Herbsleb, Mockus et al. 2000; Herbsleb, Mockus et al. 2001), while simultaneously making issues such as cultural differences more distinct (Sahay, Nicholson et al. 2003; Krishna, Sahay et al. 2004). The particular issue that I focus on in my work is knowledge (see section 2.2 for an overview of this issue in research on distributed software development), but I also attempt to understand this issue as seen in the actual everyday activities through which software development is achieved. This
Chapter 1 • Introduction

aligns with contemporary efforts in research on software development (see section 3.3 for a discussion on this), as well as distributed software development (see Appendix A for an overview of trends in this research area), where situated perspectives increasingly are complementing organisational and process perspectives in providing an understanding of the complexities of the activity.

Whereas my contribution is aimed at the information systems research area, I will draw on literature from several disciplines to achieve the objective of this thesis. For understanding the particularities of software development practice, I draw mainly on studies in the information systems, Computer-Supported Collaborative Work, and software engineering disciplines. For exploring the particular topics of knowledge in practice and change in organisations, I also draw on analyses from the organisation studies discipline, as there is much overlap in subject matter and literature in that area can provide important insights on the social aspects of organising (Orlikowski and Barley 2001).

1.4 The concept of practice

Before embarking into my research, I must also comment on my understanding of “practice” and how I am using the concept throughout this thesis.

The concept of practice has a long philosophical history and is used in many different disciplines (Schatzki, Knorr-Cetina et al. 2001). Despite a diversity of fields and approaches, there is a joint belief that “such phenomena as knowledge, meaning, human activity, science, power, language, social institutions, and historical transformation occur within and are aspects or components of the field of practice” (Schatzki 2001 pp. 2). But practice thought also encompasses multifarious and often conflicting conceptions (ibid.), and there are many discourses on, for instance, the difficult task of conceptualising the relationships between persons acting and the context in which they act (Lave 1996). While I am aware of the great body of work associated with practice theory, it is outside the scope of this thesis to delve into the nature of “practice”. I am using the term in the way that it is used within the body of literature that I draw on - where it connotes, and is a way of setting focus on, the everyday activities of people. Within this thesis, it is thus an observational lens rather than a topic of analysis in itself.

In the literature that I draw on, practice is used in reference to “the coordinated activities
of individuals and groups doing their 'real work' as it is informed by a particular organizational or group context” (Cook and Brown 1999 pp. 386-387). In this simple sense, one can make a distinction between behaviour, action, and practice. As Cook and Brown so eloquently put it (ibid. pp. 387):

*By “practice,” then, we refer to action informed by meaning drawn from a particular group context. In the simplest case, if Vance’s knee jerks, that is behavior. When Vance raps his knee with a physician’s hammer to check his reflexes, it is behavior that has meaning, and thus is what we call action. If his physician raps his knee as part of an exam, it is practice. This is because the meaning of her action comes from the organized contexts of her training and ongoing work in medicine (where it can draw on, contribute to, and be evaluated in the work of others in her field).*

Furthermore, practice is “situated”, meaning, in the words of Tellioglů & Wagner, that it “depends on context (of people, knowledge, and the nature of the task) and is cultured (in the sense of being shaped by beliefs, commitments, styles, and power relations)” (Tellioglu and Wagner 1999 pp. 76). In the context of this thesis subject, software developers have a more or less well-defined task to perform in a particular situation (working on this project, with these people, with these tools, etcetera) and, perhaps, some agreed upon or given procedure or method for doing that. But what is of interest for me as an observer are their actual activities – the “real work” - for making it happen and how that involves adhering to or digressing from objectives, procedures, methods and norms; cooperation with co-workers and other people; using available tools and resources; and so on. It is also important to remember that practice implies repetition, and so what I am talking about is not unique one-time actions, but some habitual or recurring performance.

**1.5 Thesis outline**

This section outlines briefly the content of each chapter in this thesis. An outline is also provided in Figure 1 on page 9.

*Chapter 2* reviews literature with the purpose of defining the research problem and how it will be approached. I identify an area for further research and the perspective that I assume in addressing it. Furthermore, I suggest three particular topics that will be analysed in the thesis for this purpose.
Chapter 3 details the context and methodology of my research. I present the research project that I participated in, and discuss my approach and how it aligns with current trends in the area, the empirical and analytical methods that I have utilized, and how the produced material is presented in this thesis.

Chapter 4 presents the two field studies that are the empirical base my thesis builds on. The study processes are described in detail and the results presented as a number of analytical categories that elucidate work practice in the two settings under study. This provides a first set of evidence for my thesis argument and an empirical context for further analysis of key topics.

Chapter 5 then explores the three topics suggested in chapter 2 within the particular settings elucidated in chapter 4. I do this by rendering four vignettes that illustrate episodes of change in technology, participants, and goals within the projects, and analysing the particularities of how these issues manifested.

Chapter 6 builds on the insights from the analysis of chapter 5, as I discuss what shared themes emerged across the different vignettes. This lets me characterise how changing circumstances unfold in practice as concretely shown in the two settings I study.

In Chapter 7, I discuss the results of my field studies and the findings from my analysis of changing circumstances in these settings, and present an explanation of how software developers in distributed projects deal with changing circumstances through the purposeful adaptation of practice. Seen in the light of a practice-based perspective on knowledge, this is rephrased as the development of practical knowledge and I discuss the implications of my findings in relation to previously introduced literature.

Lastly, chapter 8 summarises the results and contributions of my research. I judge how the research question has been addressed and discuss areas for further research. I also provide some personal reflections on the contents of my thesis and the implications beyond my particular topic of study.
Chapter 1 • Introduction

Chapter 2
Literature review

Chapter 3
Research context and methodology

Chapter 4
Elucidating practice through field studies

Chapter 5
Vignettes exploring issues of change

Chapter 6
Discussion of changing circumstances

Chapter 7
Discussion of knowledge development

Chapter 8
Conclusions and reflections

Research question and background

Defined research problem

Perspective and methods

Empirical analysis

Empirical analysis

Abstraction of findings

Argumentation addressing research question

Future paths

Figure 1: An overview of the structure of the thesis.
2  Issues of knowledge and change in distributed software development

2.1  Introduction

In this chapter, I identify a gap in previous research on knowledge-related issues in distributed software development. I explain how this gap can be addressed by assuming a practice-based perspective on knowledge and highlighting three particular topics that I shall explore in-depth.

A review of research in the area yields that knowledge sharing and maintenance of mutual knowledge are key issues, but also that the situated dimension of knowledge – that an important aspect of knowledge is embedded in the particular contexts where it is applied – is important to consider. Whilst issues related to the sharing of situated knowledge have been put forward in recent literature, there is still a scarcity of studies on how knowledge is developed in practice within particular contexts in distributed projects. It is towards this gap that I make an empirical contribution with my thesis.

To approach this research problem, I outline a perspective on knowledge that emphasise practice. From such a viewpoint, knowledge is not treated as something people possess, but rather as intricately tied to what they do. This entails that what people know is constantly being changed and updated as it is put into practice in ever evolving contexts. I therefore contend that, to explore how knowledge is developed in practice, one can investigate in detail how software developer groups deal with changing circumstances.

This approach corresponds to a line of inquiry into work in organisations as emerging, emphasising the importance of improvisation and viewing change as endemic to practice. I present this as a conceptual lens for studying practice and then suggest three
topics for investigating how software developers deal with changing circumstances in
the particular context of distributed software development: evolving technology,
enculturation and socialisation, and emerging project purposes and goals. These are
highlighted through a review of literature on the complexities of software development
as an activity in combination with my assumed perspective on knowledge and practice.

2.2 Knowledge-related issues in distributed software
development

In this section, I review prominent work on knowledge-related issues in distributed
software development. The purpose is to illustrate the key issues that previous research
has revealed in this domain.

2.2.1 Knowledge in software development

Knowledge is, of course, a subject in many different academic discourses concerned
with understanding socio-technical phenomena in contemporary society. It has long
been argued that knowledge creation is an essential source of competitive advantage in
today's business environment (Nonaka and Takeuchi 1995), hence establishing a
“knowledge-centric” view on the value of firms (Prusak 2001). In organisational
literature – concerned with work practices within organisations as well as the dynamics
of organisations knowledge has also become a prominent theme (Cook and Brown
1999). These notions have also led to a focus on knowledge management in literature on
collaborative software engineering. Some academics are arguing that a “software
organization’s main asset is its intellectual capital” (Rus and Lindvall 2002) and that
management of knowledge and coordination of expertise is just as important as
management of communication and coordination issues (Faraj and Sproull 2000). In
the context of globally distributed projects, the view is that the knowledge management
requirements are even more pronounced, because of the need to manage interaction
between multiple local contexts and to communicate ongoing and changing
development requirements between dispersed collaborating sites (Sahay, Nicholson et
al. 2003). Simply put, people working in distributed projects are required to coordinate
and integrate knowledge with multiple distant sources. Scholars such as DeSouza et al.
(2006) therefore argue that it is necessary to help them in this by managing knowledge
in all stages of the development process.
Such is the management view on the problem: there are knowledge-related issues and the main concern is how they can be leveraged through management efforts. But knowledge is not just a management concern and there is also much research focused on empirically investigating the nature of these issues. This research shows that knowledge is a central concept for understanding the dynamics of software development on a finer level of granularity. For example, Waterson et al. (1997) illustrated this when showing the intricate ties between distribution, communication and coordination of knowledge and expertise, and the overall organisation, progression and outcome of the project. In their case study, analysis shows how the project was gradually restructured over time less along hierarchical lines and more around the skills and expertise of individuals. This was a result of spontaneous formation of coalitions and various patterns of interaction, thus involving "a mixture of both planful and opportunistic behaviour." (ibid. p. 95). Furthermore, they argue that one of the major successes of the project was "ability to reallocate and re-negotiate tasks and responsibilities according to cognitive demands and constraints." (ibid. p. 95). This was closely tied to how information flowed in the project and the dynamics of acquisition and sharing of domain knowledge.

Waterson et al. do not analyse issues of distribution as such, although the case study exhibit some related characteristics. The project under study involved a large number of people - in the vicinity of 100 for the most part - and these participants were divided into separate teams. Interaction and coordination between them was a key issue. For instance, Waterson et al. analyses in detail communication flows and the role of individuals as "boundary spanners" and "gatekeepers". But there was no significant geographical distribution in this case and the teams themselves were flexible in the face of evolving requirements and deadlines. However, they also mention that knowledge sharing was severely limited when it came to external contractors as compared to teams within the project itself. This was due to the fact that these external contractors were "physically remote and communication channels were severely limited." (ibid. p. 92).

2.2.2 Knowledge sharing and mutual knowledge

With this we are touching on the most prominent concern of research on knowledge-related issues in distributed software development. The main problem that has been in focus is understanding how distribution affects knowledge sharing within the project. It
has been confirmed that knowledge sharing and social ties are related, and play vital roles for successful collaboration in distributed software development projects (Kotlarsky and Oshri 2005). Geographical distance between collaborators make it more difficult to have the necessary social spaces between team members where these activities can naturally take place. Kotlarsky and Oshri also mean that there is a tendency in previous literature to overemphasize the importance of technical solutions and collaborative tools for the flow and sharing of knowledge. Instead, they urge for more focus on the importance of social aspects - such as rapport - in facilitating flourishing knowledge sharing despite geographical distance.

A concept that has become central in understanding these issues is “mutual knowledge” (Cramton 2001). It refers to what is known by a person, but, more importantly, also to her awareness of who else knows it. Cramton (ibid.) advocates that a major part of the problem of knowledge sharing in distributed projects is related to building up and maintaining mutual knowledge. This is commonly referred to as a “common ground” and is considered integral to the coordination of actions (ibid. p. 346). In an experimental study of software development teams who were geographically dispersed, Cramton investigated how geographical dispersion affected the mutual knowledge problem. The empirical findings of the study highlighted a number of specific examples. One was how participants had difficulty gathering and remembering information about the context within which their distant partners worked, as well in turn communicating information about their context. Another regarded the uneven distribution of information across the teams, to a large part due to technical problems and misunderstandings, such as mails not being delivered. There were also differences in the speed at which information was acquired, as the access to the communication tools differed. Other problems were related to interpretation. The participants often made different assumptions as to the salience of different pieces of information and also had frequent problems with misinterpretation of the reasons for distant partners' silence.

The notion Cramton maintained for this study was that mutual knowledge mainly is established through three mechanisms: first-hand experience by individuals, interpersonal interaction, and through peoples assumptions about others’ knowledge on the

---

basis of social categorizations they apply to them (ibid. p. 347-349). The specific problems highlighted by the study led to a disruption of these processes. Primarily, the failures of information exchange and interpretation meant participants were more likely to make personal attributions about remote partners situations and undertakings, in turn causing collaborative disruption as it often distracted partners from fully diagnosing the problems and making modification of practices necessary to avoid re-occurrences. For our present purposes, what's interesting in the details of Cramton's findings, is how they show that the challenges of knowledge sharing in distributed projects are a result of complex social and collaborative mechanisms.

Interestingly, there are also attempts to understand the main problem in reverse. That is, on understanding what role knowledge sharing plays in alleviating the difficulties that distribution introduces. Coordination has long been identified as a key challenge in managing the complexity of software development projects (Kraut and Streeter 1995). When it comes to distributed software development, research indicates that the difficulties are even more pronounced, mainly due to the decrease or lack of informal communication and interaction among participants (Herbsleb, Mockus et al. 2000; Herbsleb, Mockus et al. 2001). Espinosa et al. (2007) have investigated the relationship between “team knowledge” and coordination in distributed software development. They discussed team knowledge primarily in terms of shared knowledge - knowledge about the team and the task held in common throughout the project - and team awareness - knowledge about what is happening in the team's task environment at any given point in time. This kind of knowledge, they explained, is key in coordination. What they were thus particularly interested in was how team knowledge related to the pronounced coordination difficulties of dispersed collaboration. Their findings showed that it was primarily shared knowledge of the team and presence awareness that helped offset some of the negative effects of geographic dispersion on coordination (ibid. p. 155-157), though the complete picture was a bit more nuanced. For example, they found that shared knowledge of the task was important for collocated members, while shared knowledge of the team was more important when it comes to the coordination between geographically distributed team members.
2.2.3 Situated and embedded knowledge

Given the distribution of participants, then, it is vital to consider in detail how knowledge is shared between the different groups. But additional difficulties exist due to the diversity of people involved in the process. Sole and Edmondson (Sole and Edmondson 2002) acknowledge the important dynamics of knowledge sharing between multiple physical sites in distributed projects. But they also highlight another inhibitive issue: knowledge of a situated nature. This issue stems in the importance of work practice. That is, how the work tasks are actually accomplished with the specific local opportunities and constraints under which they have to be performed. Knowledge about how the "real work" can be effectively accomplished is thus embedded in the work practices of a particular organisational site (ibid. p. 20). In their study, Sole and Edmondson indeed showed how "practices and knowledge differed for members of the same function located in different sites." (ibid. p. 23). But, they argued, this situated knowledge is also important for the project as a whole, as it might be useful for certain tasks in sites other than where it evolved. The problem is that sharing this situated knowledge is difficult, as there is a need to recognize and adjust for the locale-specific practices where the knowledge is embedded. In their in-depth analysis of episodes where remotely situated knowledge came to be applied, they found that conceptual misunderstandings "did not appear to result as much from differences in taken-for-granted knowledge within a function as from situated knowledge - taken-for-granted practices and information at a given site that transcended functional boundaries." (ibid. p. 30)

Similarly, Nicholson and Sahay (2004) have introduced a conceptual scheme based on the idea of embedded knowledge for understanding these issues. In their work on the subject, they describe the situation as one involving people from different sites or organisations and countries, with different languages and work practices, working together. The nature of the work is such that it covers a wide spectrum of different activities, meaning that many diverse forms of knowledge play a vital role in the process. This knowledge consists of both formal, documented types and informal, local types, and both must coincide for the success of the project. Additionally, rapidly changing technological and market conditions raise demands on both organisations and individuals with respect to acquiring and mastering new forms of both technical and
market related knowledge. Their own primary contribution to this discourse is their analysis of the nature and implications of embedded knowledge at three levels of analysis: societal, organisational and cognitive (ibid. p. 345-355). Through a longitudinal case study of an outsourcing relationship, they illustrate two examples of how the relationship was threatened due to diverging organisational and societal principles. They also discuss how these were interrelated, as organisational principles were nested within societal level structures, as well as at the organisational and individual levels.

2.2.4 Complex social and collaborative processes
The picture that is emerging is an increasingly complex one and further empirical findings help to establish it. What is being revealed is a complex reality where distribution brings “into play a multiplicity of knowledge systems, and ways of understanding and communicating them” (Imsland and Sahay 2005 p. 104). In Imsland and Sahay’s case study of a Norwegian software firm outsourcing to a Russian contractor, a number of concrete examples of knowledge-related issues were discovered (ibid.). It was shown how language problems not just are a matter of translation, but also involve difficulties related to associative words and how they relate to particular contextual conditions. They also highlighted how certain implicit assumptions are embedded in the software design itself. In their case, it took the form of country-specific business logic. A third issue was the various culturally determined ways of working, especially regarding project management hierarchies and responsibilities. This further emphasizes the contextual dependency of knowledge and how this creates problems because of the need to share knowledge across different groups of participants.

Along the same line of inquiry, Ehrlich and Chang (2006) looked at how people in global software teams acquired and leveraged the information and knowledge they needed to carry out their work. Assuming that there is knowledge exchange when people communicate, they examined this through a social network analysis. The resulting picture was a complex network of communication and knowledge exchange. People were found to be more likely to communicate regularly with local colleagues due to high awareness of each others skill, expertise and situation, but also preferred to turn to people they knew from outside the team for technical information related to
coding and programming. Ehrlich and Chang concluded that “each team member has a personal network that extends well beyond their boundaries of their current project, and is a source of valuable and important information” (ibid. p. 2). Thus we can also see that, in addition, the complexity of knowledge-related issues in distributed software development projects is not constrained within formal organisational structures.

### 2.2.5 Summary

This section has reviewed prominent work on knowledge-related issues in distributed software development. It has discussed work that is interrelated in a common narrative of what the primary problems are in this domain.

On the highest level, there are questions of how knowledge-related problems can be alleviated through management efforts. But it is also a central concern for empirical research aimed at building an understanding of the dynamics of software development on a finer level of granularity. The primary issues investigated in this body of work relate in different ways to knowledge sharing and the maintenance of mutual knowledge.

The relationship between distribution and these issues is a primary concern, but scholars have also highlighted difficulties due to the increased diversity of participants which follows with distribution. This is explained primarily with ideas of situated or embedded knowledge.

Further empirical findings help establish the image of knowledge-related issues in distributed software development as complex collaborative and social processes.

### 2.3 A practice-based perspective on knowledge

This section outlines a practice-based perspective on knowledge and explains what is of analytical interest from such a viewpoint. I also explain what I mean by the term “practical knowledge” within such a frame.

#### 2.3.1 The problem of commodification

When it comes to knowledge as a topic in research about information technology and organisations, there are several different perspectives one may assume (Blackler 1995). A general problem with much academic thought regarding knowledge-related issues in
organisations has been an underlying assumption that knowledge is something concrete that people possess (Cook and Brown 1999). Even when the notion of knowledge has been elaborated in terms of tacit and explicit (but also of local and universal, canonical and non-canonical, formal and practice, know-how and know-that) it has commonly been treated as a static thing which can be captured, stored, transformed, and transmitted (Orlikowski 2002) - also known as the problem of commodification or reification of knowledge.

Cook & Brown argue that the limitation of this implicit view of knowledge is that it distracts from the importance of accounting for the epistemic work of human action itself (Cook and Brown 1999). Consider for example the ability to ride a bicycle. Having explicit knowledge about the art of riding a bicycle alone is insufficient for acquiring the ability to ride the bike, in the sense that you cannot, for instance, transform explicit knowledge gained from someone telling you how to angle the bike in certain circumstances into the necessary tacit knowledge needed to keep the balance. In order to gain that ability you have to actually perform the act of getting on the bicycle and trying it. This tells us that the act of riding the bicycle does epistemic work itself, that human action is needed to account for knowing how to ride a bicycle. The knowledge possessed instead acts as a “tool in the service of knowing, not as something that once possessed is all that is needed to enable action or practice” (ibid. p. 388).

One alternative approach that can help researchers account for this particular role of knowledge is to focus on what people do with it in practice, how it enables them cope and achieve the work. Arguments are made for the value in a perspective that emphasize how people enact knowledge through practice, speaking of knowing as a continuous activity rather than knowledge in a static sense (Orlikowski 2002). Rather than being reified or commoditised, knowledge is then considered a process of continuous knowing indistinguishable from doing (Gherardi 2001; Nicolini, Gherardi et al. 2003).

2.3.2 An epistemology of practice and “practical knowledge”

Assuming a practice-based perspective on knowledge implies a shift in epistemology and the way we talk about the phenomenon. We move from an epistemology of possession, to an epistemology of practice (Cook and Brown 1999).

In conventional assumptions about the nature of knowledge, there are many different
kinds at work. One basic conceptualisation is Polanyi’s (Polanyi 1983) distinction between explicit (that can be articulated) and tacit knowledge (that is implicit in what we can do). For instance, Nonaka and Takeuchi (Nonaka and Takeuchi 1995) argue that knowledge creation is achieved through a dialogue between tacit and explicit knowledge, where the different kinds are converted back and forth in various processes.\textsuperscript{4} Another common distinction is that between individual and group knowledge. For instance, one can discern two distinct approaches to organisational memory along those lines, one focusing on individual cognitive capabilities, and the other on the cognitive property of an organisational collective entity (Bannon and Kutti 1996). Blackler (Blackler 1995) has also identified a number of different images of knowledge given in organisation studies: embrained, embodied, encultured, embedded, and encoded knowledge. In the context of work practice, another important concept that is that of contextual knowledge, knowledge that becomes relevant only in light of particular problems at hand (Barley 1996).

However, a practice-based perspective on knowledge transcends definitions of different kinds of knowledge. It is a perspective that “that focuses on the knowledgeability of action, that is on knowing (a verb connoting action, doing, practice) rather than knowledge (a noun connoting things, elements, facts, processes, dispositions).” (Orlikowski 2002, pp. 250-251). That is, assuming this point of view, we talk only about what people know, and only in terms of what it is they do. In previous research on knowledge in distributed software development, there is a tendency to talk about different kinds of knowledge. Both Nicholson & Sahay (2004) and Sole & Edmondson (2002), for instance, speak of situated or embedded knowledge as something distinct. In this thesis, I move away from such distinctions. I use the term “practical knowledge” to articulate the issue I am addressing. However, this does not refer to any particular kind of knowledge, but to what the software developers know about how the work can be done within particular context. It is the knowledgeability that enables them to participate in practice as observable in what they do.

\textsuperscript{4} It should be acknowledged that Polyani himself, while making a distinction between explicit and tacit, also maintained that they were indistinguishable, as one could not exist without the other. This nuance of his original idea is often left behind in contemporary use of the concepts.
2.3.3 Practical knowledge as inescapably flexible and evolving

This practice-based approach to knowledge and knowing in organisations has its origins among learning theorists who have attempted to explain the relationship between learning and practice (Lave and Wenger 1991; Chaiklin and Lave 1993; Wenger 1998). In this understanding, knowledge is not seen as something that can be packaged and consequently transferred between people. Rather, it is an ability that is continuously learned. Learning itself is seen primarily as the social process of a learner becoming a practitioner (Lave and Wenger 1991) and participating in a community of practice (Wenger 1998).

The key aspect of this perspective for our current purposes is its emphasis on the inseparability of knowledge, learning and practice (Brown and Duguid 1991). Assuming a view of knowing as constantly enacted in practice makes it an ongoing and situated activity. Both activity and context are inescapably flexible and evolving (a notion further discussed in section 2.5.1), entailing constant changes in both knowledge and action (Chaiklin and Lave 1993). Shifting focus from knowledge as something people have to knowing as something they do means that we are now talking about a phenomenon that “(a) manifest in systems of language, technology, collaboration and control (i.e. it is mediated); (b) located in time and space and specific to particular contexts (i.e. it is situated); (c) constructed and constantly developing (i.e. it is provisional); and (d) purposive and object-oriented (i.e. it is pragmatic).” (Blackler 1995, pp. 1039). Two concepts will help us elaborate on what this entails for understanding knowledge-related issues.

*Learning in practice*

Thus, from this view it seems like learning is an inherent aspect of everyday practice. In the words of Lave: “It is difficult, when looking closely at everyday activity […], to avoid the conclusion that learning is ubiquitous in ongoing activity, though often unrecognized as such” (Chaiklin and Lave 1993). But what do we mean by learning in this context? What we are talking about is a sort of learning that is embedded in everyday practice and serves as an essential mechanism for people to accomplish their tasks. This means learning the general and contextual specifics of the occupation, but, as pointed out by Duguid (2005), it is “impossible to specify and hence codify all the knowledge involved even in the most elementary practice”. Thus, we are not speaking
of learning in the sense of exclusively acquiring codified knowledge.

Brown, Collins, et al. (Brown, Collins et al. 1989) introduce a set of tools as a metaphor for how knowledge is enacted in practice and use this to further elaborate on the meaning of learning in practice. Rather than just acquiring the tools, it is through using them that people gain the ability to apply them properly in an activity. But appropriate use is a function of the culture in which the tools have been developed and the accumulated insights of the community where they are used. Therefore, it is not possible to learn to use a tool appropriately without understanding the culture and community in which it is used. In other words, “using them entails both changing the user’s view of the world and adopting the belief system of the culture in which they are used” (ibid.). Therefore, learning in practice, seen as an essential mechanism for coping with an ever changing environment and accomplishing work, is the result of a person acting and interacting within a community of practice. In this view, learning in practice is about becoming a practitioner, not learning about practice (ibid.).

**Communities of practice**

Looking closer at the concept of “communities of practice” helps us understand what learning to participate means. Further elaborated by Wenger, the concept attempts to associate to the way that practice is the source of coherence in communities (Wenger 1998). This can be described as a relationship along three dimensions: a joint enterprise, a mutual engagement, and a shared repertoire. Someone who is a competent participant has the ability to engage with other members and respond to their actions, to understand the enterprise of the community deeply enough to take some responsibility for it and contribute its continuous negotiation, and to make use of the repertoire to actually engage in the practice and with other members (ibid.).

There are a number of perspectives that offer explanations of the actual process through which people achieve this in real-world contexts (Brown and Duguid 1991). Lave & Wenger’s original concept of legitimate peripheral participation portrays it as a process in which mastery of knowledge and skill requires learners to move from the periphery toward full participation in the sociocultural practices of the community (Lave and Wenger 1991).
2.3.4 Summary

This section has outlined a perspective on knowledge that emphasize practice. To avoid the problem of commodification, knowledge can be viewed as a continuous activity, as something that constitutes a person's ability to engage in a practice.

Looked at from this perspective, knowledge is also constantly being changed and updated as it is put into practice in flexible and continuously evolving contexts. One of the consequences of assuming such a perspective is thus that there is value in analysing how knowledge is enacted and developed in practice.

This point of view also entails a change in the way we talk about knowledge. In this thesis, I use the term “practical knowledge”, not in reference to a special kind of knowledge, but to what the software developers know about how the work can be done in particular contexts as observable in what they do.

Lastly, our attention is also drawn towards matters of enculturation and socialisation, as knowledge in this sense is about a person's ability to participate in a community of practice.

2.4 The need for investigations of how knowledge is developed through practice

This section uses the insights from the review of previous research on knowledge-related issues and of literature on a practice-based perspective on knowledge to identify an area for further research.

2.4.1 The need for in-depth studies of knowledge-related issues in practice

The review of literature on knowledge-related issues in distributed software development illustrated how they are a matter of complex collaborative and social processes. However, it remains a fact that most of the previous research has centred on mechanisms of knowledge sharing between, and maintenance of mutual knowledge among, software developer groups in various geographically distributed locations. While being an important aspect, I argue that this is only one side of the problem. The other important side is related to actual work practice in particular contexts within the projects.
In the review, I also highlighted how the situated aspect of knowledge has been put forward, as researchers have shown how much of software developers knowledge is intricately tied to how the work tasks are actually accomplished with specific local opportunities and constraints (Sole and Edmondson 2002). Much knowledge that is of importance for the project is thus embedded in local work practices and this makes it more difficult to share between distant sites (Nicholson and Sahay 2004). This connects with the practice-based perspective of knowledge, which also emphasises the intricate ties between knowledge and practice. What is of particular interest from such a perspective is knowledge as seen in a persons ability to participate in practice and how this ability is constantly enacted and developed in particular contexts.

However, although issues of knowledge sharing in relation to situatedness or embeddedness have been investigated in recent literature (Sole and Edmondson 2002; Nicholson and Sahay 2004; Imsland and Sahay 2005), there is still a scarcity of in-depth empirical studies of the ways that software developers knowledge is continuously maintained in practice in distributed software development settings. Partly, this is a methodological insufficiency, as most previous studies rely primarily on retrospective accounts of practice. In light of these insights, I argue that there is a need for longitudinal, observational studies of how knowledge is enacted and developed in ongoing practice within distributed software development projects. Such studies would increase our understanding of the underlying conditions for how knowledge sharing can actually happen in such settings, and this is indeed the purpose of my thesis.

2.4.2 The development of practical knowledge through dealing changing circumstances

As for what such an investigation of knowledge in practice should focus on, this thesis takes its cue from the perspective outlined in section 2.3. Assuming this view entails that focus should be put, not on knowledge as something people have, but on knowing as something they do. This is knowledge as the ability of individuals to participate when the group perform work with particular local opportunities and constraints, that is, in particular circumstances. Since it is intricately tied to the particular circumstances where it is applied, the development of this knowledge will also come in response to changes.

---

5 A partial exception is Sole and Edmondson (2002), who's case study triangulated different data sources, including observation of practice.
of these circumstances. Activity and context are constantly evolving together, but there is still a lack of studies of how this co-evolution happens during ongoing practice in distributed software development.

Thus, I translate the need for further empirical research on how knowledge is enacted and developed in practice, into a need for studying how change in circumstances unfolds and is dealt with in practice. I particularly focus on practical knowledge - referring to what software developers know about how the work can be done in particular contexts - and how this is continually developed as they deal with changing circumstances. Doing so represents a new line of inquiry into knowledge-related issues in distributed software development, as the connection between knowledge and change is previously unexplored. This also contributes towards our understanding of the complexities of distributed software development, as the ever evolving environment in which the work has to be accomplished is an aspect not often acknowledged.

2.4.3 Summary

To summarise the argument of this section, I am proposing that, to further our understanding of knowledge-related issues in distributed software development, what is needed are in-depth studies of how practical knowledge – that is, what software developers know about how the job can be done in particular contexts - is enacted and developed in practice. One way of looking at this is by focusing on how change of circumstances unfold and is dealt with, as practical knowledge is intricately tied to the evolving context where it is applied. Thus, I address the issue of knowledge in this thesis, not as an immaterial object that is “knowledge” or in terms of how individual software developers “learn”, but as seen in the actual activities of software developers as they deal with changing circumstances. The contribution of this effort is to unearth the important dynamics of how software developers continually develop their practical knowledge within particular contexts of distributed software development projects.

2.5 Topics for studying changing circumstances in distributed software development practice

The focus of this thesis, then, is set on the development of practical knowledge as seen in how software developer groups encounter and deal with change of circumstances in practice. What we need now, is to de-construct this focus into a usable conceptual lens
which can guide the investigation towards concrete, observable issues. This is the main purpose of this section. It begins by considering what characterises a view on work practice that focuses on change. By drawing on literature about the complexities of software development activities, it then suggests three particular topics for investigating change in the context of distributed software development practice.

2.5.1 Improvisation, emergence and change – a perspective on practice

Focusing on issues of change corresponds with a line of inquiry in organisational studies that emphasize the importance of impromptu activities and how organisational workers deal with the unforeseen and unexpected (Weick 1998). This is closely connected to processes of creativity and innovation, and improvisation has been suggested as a concept which can help in understanding these mechanisms (ibid.). In Weick's portrayal, inspired by improvisation in jazz music, improvisation as a mindset for organisational analysis helps characterising the tension involved in mixing the intended and the emergent. Improvisation “involves reworking precomposed material and designs in relation to unanticipated ideas conceived, shaped, and transformed under the special conditions of performance, thereby adding unique features to every creation” (Berliner 1994 p. 241, cited by Weick 1998 p. 544).

This is an alternative way of understanding how people work in organisations. What is of interest from such a perspective is the situated performance where thinking and action emerge simultaneously and in the spur of the moment (Ciborra 1999). Continuing to use improvisation in music as a model, Ciborra writes that, to “improvise effectively, the musician knows the conventions and the rules of a given musical style, since they provide a sort of library of musical resources that can be recombined to generate new instances of the score.” (ibid. p. 79). In terms of organisational workers, this means that they know a repertoire of routines, methods, standards, norms and conventions - ways of doing things within the organisation. But this repertoire has to be applied in particular situations which are not clear-cut or easily interpretable. Hence, what they do can be described as improvising using previous resources under new circumstances to create new action (music).

This also has implications for the organisation. Orlikowski (1996) has proposed an understanding of organisational transformation as the result of micro-level changes. In
contrast to models based on notions of planned change, technological imperatives, or punctuated equilibrium, this perspective emphasizes how organisational actors subtle shifts in action over time transforms work practices, organizing structures, and coordination mechanisms. What Orlikowski highlights is that, in addition to deliberate strategies (Mintzberg and Waters 1985) as the source of change, there is also emergent change as new patterns of organizing are realized in action (Orlikowski 1996 p. 65). This emergent change is not anticipated or planned, but a posteriori, happening without explicit intentions as organisational actors respond to everyday contingencies, breakdowns, exceptions, opportunities, and other unintended consequences. The idea is that organisational transformation also can be realized through the ongoing variations that emerge from these "slippages and improvisations of everyday activity" (ibid. p. 89).

A related topic that has to be mentioned here is Hutchin's work on distributed cognition (Hutchins 1991; Hutchins 1995a; Hutchins 1995b). Hutchin's speaks of intricately structured and tightly coordinated groups in terms of information-processing computational systems, and puts forward ideas of how work is continuously organised by adaptation. However, the perspective is different from the practice-based approaches that I draw on here, as it is more about cognitive issues and the adaptive capabilities of the organisations themselves, than about how the actors cope with contingencies and unexpected events in everyday work practice. (I will return briefly to this distinction in my discussion in chapter 7.)

The perspective on practice that appears in the work of Weick, Ciborra, and Orlikowski is connected to two broader schools of thought. First, these ideas are linked to a perspective on organisations that emphasizes action rather than stability. From this perspective, organisations are enacted through the ongoing action of organisational members (Giddens 1984). Organisations are not seen primarily as the outcome of hierarchical and function task decomposition, but equally as ongoing processes of in situ sense-making and decisions (Weick 1995; Ciborra 1999). It is outside the scope of this thesis to delve further into these theories, but they carry similarities to the portrayal of distributed software development projects as work environments in constant flux offered by this thesis. (See Ciborra (1999) for more elaboration on how shifts in thinking about organisations and organising connects with the ideas of improvisation and situated change drawn on here.)
Secondly, the emphasis on improvisation and change connects with ideas regarding the importance of work practice in organisations (Barley 1996). Orr's (1996) influential study of the work practices of a group of service technicians revealed a complexity in practice that diverged considerably from both assumptions regarding technicians work and espoused practice. Furthermore, he showed how this complexity constrained how the work could be done and that it thus had crucial implications for this making policy about the work. These insights have important methodological implications for the research presented in this thesis, as accessing the actual work practices of software developers in distributed projects becomes a key concern. This is further elaborated on in chapter 3.

2.5.2 Prominent topics of change in distributed software development

The question that remains, is what concretely to look at when studying “changing circumstances”. This sub-section highlights three particular topics related to change and argues that these are relevant for investigating changing circumstances in the context of distributed software development practice.

2.5.2.1 Evolving technology

In layman's terms, the computer industry is often characterized as rapidly evolving, constantly changing, never stable. The concept of Moore's Law - the idea that computer technology develops exponentially – has, even though it originally refers specifically to the rapid development of computer hardware, come to represent a common view on the industry as a whole, including the software side of it. The notion of a constantly changing environment has been a subject of interest in academic discourse as well. Eisenhardt (1989) spoke early of the microcomputer industry as a “high-velocity environment”, defined by rapid and discontinuous changes in demand, competition, and technology. One of the consequences of this situation, she stipulated, was that the information available for organisational decision makers often was inaccurate, unavailable or obsolete. In her studies of how this condition affected strategic decision making in organisations, she found that decision makers actually used less information, while simultaneously developing more alternative options. Continuing along these lines, Brown and Eisenhardt (1997) later extended the ideas about perpetual change to apply even to the firms themselves. They argued that firms in high-velocity industries such as
Chapter 2 • Issues of knowledge and change in distributed software development

the microcomputer one – characterized by “short product cycles and rapidly shifting competitive landscapes” (ibid. p. 1) – compete by changing continuously. Their organisational structures and processes possess certain features that enable this constant change.

The development of technology itself is central in these ideas about what defines the computer industry. It was, for example, at the heart of Eisenhardt’s analysis, as she cited substantial technological change as one of the key sources of volatility (Eisenhardt 1989 p. 544). The particular question of how organisational routines and behaviours are affected and developed by the implementation of new technologies and applications has been important in organisation studies (e.g. Edmondson, Bohmer et al. 2001). In information systems research, understanding the role of information technology in organisations is a central question (Avgerou 2000) and the close link between technological change and organisational change is also strongly emphasised (Orlikowski 1992). Consequently, empirical studies of information systems development (another central issue in the domain) have highlighted technology change as a significant topic (Orlikowski 1993). When it comes to distributed software development, the phenomenon has come to play a significant role in today's software development industry (Sahay, Nicholson et al. 2003; Carmel and Tjia 2005). But it has also been enabled, and is to a large extent mediated, by the rapid development of information and software technology. Thus, work in these settings is both contributing to the evolution of technology, and being intimately affected by it. Indeed, the issue of rapidly changing technological circumstances has been highlighted by researchers of this phenomenon (Nicholson and Sahay 2004).

When Orlikowski (1996) suggested her situated change perspective on organisational transformation (introduced in section 2.5.1 above), it was backed up with an empirical study of how a customer support department in a larger software company changed their organizing practices and structures over a period of 2 years. These changes were related to the introduction and implementation of a new incident tracking support system. What followed the introduction were "ongoing, gradual, and reciprocal adjustments, accommodations, and improvisations" (ibid. p 89) enacted by the actors as they "attempted to make sense of and appropriate the new technology and its embedded constraints and enablements" (ibid. p. 89). It is important to point out that there is no
argument here that technology caused these changes. Rather, it enabled them. But it does show us the intricate relationship between technological changes and significant organisational and practical changes.

Lanzara expressed this nicely when he wrote that when “a new computer-based technology enters and organisation or an established domain of practice a whole range of different phenomena can happen, depending on the nature of the technology and on the cognitive, social and institutional features of the setting or the domain.” (Lanzara 1999 p. 333). He primarily emphasises the complex game that starts among the players when they encounter and cope with the introduction of novel technology. This game is dominated by a tension between change and stability, as the technology is appropriated into existing practices and features. Technology has structuring properties, but is also sensitive to the peculiar features of the hosting environment and institutions. It may also reshape the basis for competence and practical knowledge of people at work.

In this thesis, technological change is chosen as one significant topic for investigating changing circumstances because of three reasons that have been highlighted above. First, its pervasive role in the phenomenon under study. Second, the fast-paced development of technology even as it is in use. And third, how technological evolution is intricately tied to changes in organisation and practice.

2.5.2.2 Enculturation and socialisation

The practice-based perspective on knowledge assumed in this thesis has already drawn our attention towards issues of participation. When studying knowledge seen as something that constitutes an individuals ability to engage within a community of practice, learning in practice becomes a central issue (Brown and Duguid 1991). Someone who is a competent participant has the ability to engage with other members and appropriately respond to their actions, to understand the shared enterprise deeply enough to contribute to ongoing negotiation, and to make use of a repertoire of tools and practices specific to the community (Wenger 1998). Becoming a competent participant is the result of acting and interacting within the community of practice. It is about forming one's identity within the community through socialisation (Wenger 1998) and becoming a practitioner, something that can be described as a process of enculturation (Brown and Duguid 1991).
The importance of such issues can also be seen in the practical problems of the software development domain itself. The demands on software projects tend to fluctuate. Often, the response to such fluctuations is to assign new people. But, as Brooks (1995) so eloquently put it, men and months are not interchangeable commodities. Whenever people join new projects, Brooks writes, there are added burdens of communication made up of two parts: training and coordination. Sim and Holt (1998) have called this the “ramp-up problem” of software projects. Joining a new project requires software developers to learn about a new problem domain and software system, but also adapt to the new working environment (ibid.). In their case study, Sim and Holt call this a naturalisation process. Their findings indicate that documentations and tools are not sufficient for facilitating this process. What is more important are the organisational processes that surround the arrival and integration of new developers to the team (ibid. p. 370).

When it comes to open source software development – which is one of the cases investigated in this thesis – the importance of socialisation and enculturation is even more exacerbated. At the most basic level, there is a correlation between the size of the community and the success of the project (Krogh, Spaeth et al. 2003). Open source projects usually consist of a small core group who writes most of the key functionality, while there is a much larger group of peripheral participants who mainly report and fix bugs (Mockus, Fielding et al. 2002). But because they are volunteer-based, open source projects commonly have a high turnover rate. This is especially true in the peripheral group, but also applies for the core members, who are vital because of their key knowledge about the system (Krogh, Spaeth et al. 2003). If a project cannot attract and retain new contributors - both peripheral developers and core members - it is not likely to be successful in the long run (Ducheneaut 2005). As expressed by Ye, Nakakoji, et al. (2005), “for OSS [Open Source Software] projects to sustain, their communities have to be able to regenerate themselves through the contributions of their members and the emergence of new Core Members and Active Developers.” In the light of this, Ducheneaut has argued for the importance of understanding the socialisation of newcomers, how “individuals external to the community are progressively socialized (or not) into a project.” (Ducheneaut 2005) In the same vein of argument, Edwards propose theories of situated learning and the related idea of legitimate peripheral
participation as analytical tools for understanding the process of socialisation in open source communities (Edwards 2001).

What we see, then, is that the coming and going of members are a common ingredient of everyday work in software development projects. The perspective assumed in this thesis also give priority to matters of participation. Thus, issues related to socialisation and enculturation of new members are chosen as another topic for investigating changing circumstances in this thesis.

2.5.2.3 Emerging project purposes and goals

The last aspect we shall look to is the development process itself, something that has been the subject in a vast array of academic writing. What I am interested in, though, is not software development methodologies, but rather the nature of how software development progresses as an actual activity. In one influential foray into this subject, Lanzara (1999) has highlighted the role of transient constructs in system development projects. Transient constructs are temporary tools that are created or assembled for the purpose of achieving larger goals. It is the scaffolding put in place during the building of a house, for example. Lanzara takes this concept to explain the complex, emergent features of design in practice. When systems are designed and developed, there is constantly transient knowledge that the actors need to learn or create as a means of moving forward. Lanzara suggests *bricolage* as an appropriate strategy for this process, as it involves construction with whatever materials are at hand. This is because systems are not implemented from scratch. "Designers start building up with the materials they have at hand. So, they start small and rely on pre-existing arrangements. A large part of designing consists of transforming and reshaping what is already in use, or creatively rearranging components." (ibid. p. 346).

In illustrating the role of transient constructs in design processes, Lanzara also conveys an image of the development process itself that is informative for us. He writes: "Close-to-action observation and on-line tracking of design and change processes in organisational and professional settings reveal that such processes happen in ways that remarkably differ from the orderly, structured paths that most current theories and methods tend to assume." (ibid. p. 334). The process is an emerging one, where agents continuously re-adapt and reinvent systems features and functionality in accordance
with changing environmental and personal requirements (ibid. p. 332).

This work builds on Ciborra and Lanzara's (1994) earlier notions of "designing-in-action" and "formative contexts", and the argument is also consistent with more recent empirical research into systems development processes in practice. For example, Luna-Reyes et al. (2005) case study portrayed information systems development as "a socio-technical phenomenon embedded in an emergent process of change in professional practice." (ibid. p. 94). As a complement to linear representations, they suggested a recursive model of the process, one that can capture the emergent and iterative nature of systems development.

These insights about the iterative, emergent nature of software development in practice must also be put in relation to a contemporary movement among practitioners, namely the agile software development movement. Agile methods are becoming very common and are increasingly adopted throughout the industry. Therefore, this movement is having an impact on how software is developed (Dybå and Dingsøyr 2008). At large, it is a reaction by experienced practitioners against plan-based software engineering approaches. The agile manifesto\(^6\) was established in 2001 by a group of representatives of various burgeoning software methodologies. It outlines a set of 12 principles\(^7\) that underpins the idea of agile software development. Two of the key principles which are relevant for our present purposes regard the importance of reacting to feedback and inevitably changing requirements during the whole development process, and of allowing for iterative, emerging development to achieve the best architectures, requirements and designs (see Fowler and Highsmith (2001) – two of the original authors of the manifesto - for more elaboration on these principles).

Thus, insights from both academic writings and contemporary movements among practitioners make it clear that emerging objectives and requirements are a natural part of software development projects. Whatever the intentions and settings, a projects purposes and goals are not entirely set from the start, but rather are in a constant flux throughout the projects. This is the reason why this thesis chooses this issue as a third topic for investigating changing circumstances.

\(^7\) [http://agilemanifesto.org/principles.html](http://agilemanifesto.org/principles.html) (Accessed: 2010-01-21)
2.5.3 Summary

Section 2.5 has delved into issues of change. First it characterised what a focus on change entails in understanding organisations and practice. It made links to a view on organisations as enacted, and also to a methodological perspective which emphasise the importance of non-canonical work practice.

It then suggested three topics related to change that are prominent in the context of software development as an activity: evolving technology, enculturation and socialisation of new members, and emerging project purposes and goals. These are relevant topics of investigating how practical knowledge is developed as software developers deal with changing in circumstances.

2.6 Chapter summary and conclusions

In this chapter, I first reviewed research on knowledge-related issues in distributed software development, showing that an area for further research is how practical knowledge is enacted and developed in practice within particular contexts. By assuming a practice-based perspective on knowledge, I argued that this can be achieved by studying how software developer groups in distributed projects encounter and deal with changing circumstances. This is because, from such a point of view, practical knowledge is intricately tied to the ever-evolving context in which it is applied.

I discussed what this focus on change entails with the help of a line of inquiry in organisation studies that emphasise emergence and improvisation. To provide concrete issues for investigating changing circumstances in distributed software development, I suggested three topics. First, evolving technology, due to its pervasiveness to the phenomenon under study, constant evolution even as it is in use, and intricate ties to both changes in organisation and practice. Second, enculturation and socialisation, as the assumed perspective on knowledge emphasises matters of participation and software development projects are characterised by fluctuating membership rosters. Third, emerging project purposes and goals, as both academic writing and contemporary movements among practitioners illustrate how a projects objectives are in a constant flux.

Thus, to make a contribution to research on knowledge-related issues in distributed software development, I will examine in detail these three topics in particular settings.
What I find on how the software developers cope with such issues will be reinterpreted as the development of practical knowledge.
3 A qualitative investigation of software developers work – methodology and research methods

3.1 Introduction
In the previous chapter, I set focus on how software developers in distributed projects develop their practical knowledge as they deal with changing circumstances and suggested three topics for exploring how this happens. This was coupled with a lens that gives priority to non-canonical work practice, or the actual everyday activities of software developers in distributed settings. This chapter presents the methodology of my research, describing the research context and perspective, as well as the empirical and analytical methods employed in order to investigate those practices. This methodological framework has been applied in two field studies, which are presented in detail in chapter 4.

My research took place within the socGSD project at the University of Limerick. It has inherited the projects focus on social, cultural, and organisational aspects of distributed software development. Part of this background is also the projects methodological assumptions, stemming in ethnographically-informed studies of software development work, and work practice studies in the computer-supported collaborative work and organisation studies disciplines.

I discuss how this perspective aligns with a stream in studies of software development activities where social aspects are increasingly emphasised and qualitative, interpretive approaches becoming more common. The potential of these approaches is that they can help researchers elucidate actual practice with the purpose of informing practice,
technology design, or further research, making them appropriate for my thesis research. However, unveiling actual work practice also represents a significant challenge. I describe participant observation and grounded theory coding as appropriate methods for coping with this challenge. Lastly, the analytical strategy of the thesis itself – that is, how the empirical material has been worked up into a coherent thesis – is presented. It aims to providing thick descriptions of practice, characterising the topics investigated by focusing on how episodes that exemplify the three suggested topics (introduced in section 2.5.2) unfolded in practice.

### 3.2 Research context and methodological background

This section provides information about the research project – both in terms of its participants and goals, and in terms of methodology - within which my research took place.

#### 3.2.1 The socGSD project and group

The research work undertaken for this thesis was part of a research project focused on work practices and collaboration in globally distributed software development projects. The project was called Social, Organisational, and Cultural aspects of Global Software Development - using socGSD as its acronym - and was a joint project between the LERO (The Irish Software Engineering Research Centre) and the Interaction Design Centre at the University of Limerick, Ireland. The scope included all kinds of distributed software development work, from formal projects in multinational corporations, to ad-hoc, free open source projects.

The research group who worked on the project was both multidisciplinary and international. Throughout the project, the group composition varied, but for the most part it had 5 to 7 members. This number includes 1 professor, 2 post-doctoral fellows, 2 students (PhD/MSc), 1 research assistant, and often 1 to 2 advisers. Member backgrounds included a variety of academic domains: the computer science disciplines of software engineering, information systems, human-computer interaction and computer-supported collaborative work, but also psychology and sociology. Most had also worked in interdisciplinary institutions before. This meant that the group was

---

8 One of the LERO cluster projects, funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI).
familiar with a multiplicity of academic perspectives. A large part of the group also had backgrounds related specifically to software development. For example, two members had significant experience from industry, while I have a background in systems development, both from education and as a practitioner. Nationalities of the group included Irish, British, Romanian, and Ghanaian, as well as myself, who am Swedish. This diversity added to our understanding of cultural issues.

All research work undertaken in the socGSD project was collaborative, with all members contributing with their special competence to the various efforts. However, the members also represented individual strands of research in the project, corresponding to their own research interest. This means that my research for this thesis – while benefiting from the great support of the group – is one of the individual branches of the socGSD project.

3.2.2 Focus and methodology of the socGSD project

The focus of the socGSD project was on social, organisational and cultural aspects of distributed software development. In particular, we in the research group were studying the phenomenon as a social activity rather than a formal process or technological challenge. Our research approach in the project was based on two strands of methodological thought: ethnographically-informed empirical studies of software engineering (Dittrich, John et al. 2007; Robinson, Segal et al. 2007), and work practice studies in the fields of computer-supported collaborative work (CSCW) (Heath, Hindmarsh et al. 2000) and organisational studies (Barley 1996). This means that we were interested in investigating the actual work practices – as compared to canonical and espoused practices – of people in distributed software development projects, and in studying this socio-technical phenomenon in its natural state. The naturalistic emphasis was an important pillar in our research approach. Utilizing such an approach traditionally means not relying solely on “artificial” settings, like experiments or interviews, but instead drawing on participant observation and in situ conversations as the main source of information.

Another important pillar in our methodology was the concept of “context”. The naturalistic research tradition emphasizes that human actions are based on social meaning – intentions, motives, beliefs, rules, values, and so on (Hammersley and
Atkinson 1996). In order to be able to account for these aspects, studies of situated everyday practice must not separate persons acting from the social world of activity (Chaiklin and Lave 1993). Hence, accounting for the context in which the observed activity is situated, and for the relations between the persons acting and the social world, was another important endeavour in our studies of work practice.

**3.2.3 Summary**

My research took place within the socGSD project, which focused on actual work practices in distributed software development projects. The methodology is made up of two pillars – a naturalistic approach and an emphasis on context – extending from a foundation of ethnographically informed empirical studies of work practice.

The multidisciplinary research group provided a wide array of competence in academic perspectives and methods. But although the research within the project were collaborative efforts, the different members of the research group represented different strands. My research for this thesis has inherited the general focus and methodological background of the socGSD project, but also represents one distinct branch of it.

**3.3 Research perspective**

This section examines some relevant qualitative perspectives that have been employed in previous research on software development work. It shows how the perspective of this thesis research aligns with a contemporary stream in the domain and discusses what the potential contributions of such an approach are.

**3.3.1 Qualitative perspectives on software developers**

There are a handful of historical examples where social activities surrounding software development work processes were the focal point of research. Over two decades ago, Curtis, Krasner, et al. (1988) argued for the importance of understanding how human and organisational factors affect the execution of software engineering tasks. They set out to investigate “the processes and mechanisms through which productivity and quality factors operate, rather than developing a qualitative assessment of their impact.” (ibid. p. 1269). Another prominent early example is Walz, Elam, et al. (1993), who studied how software developers built and maintained shared mental models during design activities. They examined how the activities unfolded over time inside an actual
design team. However, even though the interest in software processes arose during the late 1980's (Boehm 2006), such examples have been an exception in the main software engineering research community. Only during the last decade has there been a broader trend towards an emphasis on social, cultural and organisational aspects of software development (Herbsleb 2005). What we are also seeing today is qualitative research of collaborate software development being performed within several different research communities. Such perspectives are taken in the software engineering community, but also, for example, by researchers of computer-supported collaborative work, systems development and agile methodologies (Dittrich, John et al. 2007).

In terms of methods, Curtis, Krasner, et al. (1988) were doing qualitative data generation in the form of interviews and their analysis was also qualitative in nature. Walz, Elam, et al. (1993) focused on actual activities at design meetings, a number of which were video recorded and analysed at a later stage. However, while these two studies were based on qualitative data, they were utilizing few sources and also restricting themselves to looking at a subset of the software activities. The trend today is towards more holistic approaches. Focus is broader, as what is of interest is the actual, situated and everyday activities through which software developers accomplish their work. In order to investigate these social activities, researchers adopt qualitative research strategies that cover logic research design, data generation techniques, and analytical methods (for discussion of what research strategies includes see Yin 2003). In particular, the researchers utilize many different sources to access the activities investigated, combining observation of practice, interviews, discussions with practitioners, and collection and analysis of artefacts (see Holmström & Henfridsson (2006) for a good illustration of how such a multiple source approach can be utilized).

Interpretive case studies (Klein and Myers 1999) are thus becoming more prominent in the area. Ethnographically inspired approaches have become particularly popular for investigating software development as a social and cultural activity (Robinson, Segal et al. 2007). In essence, an ethnography is a “thick” description of a setting and of the actors in it, an account that is sufficiently detailed to give an understanding of the culture under study, what the world looks like to the people who inhabit it (Hammersley and Atkinson 1996). Whereas many case studies may rely on interviews supplemented by documentary evidence as the primary data source (Yin 2003), an ethnographic study
relies primarily on detailed observational data, complemented by interviews and collected documents (Myers 1999).

The ethnographic approach originated in anthropology and has also played a big role in sociological research during the last half century or so. From there it has been spreading throughout the academic world and is now common in many fields, including computer science disciplines. But computer scientists, rather than doing full ethnographies – meaning research that result in an ethnography: “a theorized account of the culture studied with ethnographic methods” (Delamont 2007) – are adopting the ethnographical principle of spending time in the field observing the concrete activities which are the subject of research.

Another strand of social inquiry related to the increased empirical focus in the study of software development is grounded theory approaches. The core of these approaches is their bottom-up way of “generating theory through research data rather than testing ideas formulated in advance of data collection and analysis” (Dey 2007 p. 80). Outside of the social sciences it appears primarily as a form of data analysis. Nevertheless, this methodology also strongly encourage the emphasis on empirical material and social activities, and both of these approaches play key roles in this current methodological trend in software development studies.

3.3.2 The potentials of qualitative approaches

There is, then, an increasing focus on the social and situated aspects of software development, coupled with research approaches that build on extensive empirical studies of actual, real-life activities. But what are the specific contributions to the research communities of such perspectives and approaches? We can discern this by looking at a few prominent examples.

One example of an ethnographically inspired study of software developers is Sharp and Robinson’s (2004) study of the practice of the agile development method eXtreme Programming (XP) in a small software company. In this study, five characterizing themes that underpinned the work of software developers applying XP were distinguished. This shows the potential of participant observation to make the elusive social practices available for discussion and support. Another example of a qualitative study focusing on actual practices is the Singer et al. (1997) study of daily activities of
software developers in a team maintaining and extending a large telecommunications system. The purpose of this study was specifically to inform the design of tools for software engineers and the analysis of actual work practices was used to derive a number of specific requirements for such a tool.

Recently there has also been a rise in studies of the complexities of distributed software development projects based on extensive field studies. Participant observation and grounded theory coding techniques are common methods. Observations combined with interviews and grounded theory analysis helped Damian, Izquierdo, et al. (2007) reveal how different organisational cultures manifested in the way team processes were followed and the use of common tools. They found that these cultures were an important factor for how awareness of change was promoted in a distributed project. Similarly, Boden, Nett, et al. (Boden, Nett et al. 2007) studied coordination practices in distributed software development projects of small to medium sized companies by participant observation. This helped them highlight how “the evolving complexity of articulation work in the context of distributed projects was partly unexpected by the actors beforehand” (ibid.). They found that the formal plans did not describe the actual work practice that were actually emerging through continuous renegotiation of task allocation. (Appendix A elaborates further on the trends in research on distributed software development.)

<table>
<thead>
<tr>
<th>Areas of contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Making practice available for inspection, comment, and critique.</td>
</tr>
<tr>
<td>2 Informing design of tools, techniques, and methodologies for supporting practice.</td>
</tr>
<tr>
<td>3 Elucidating previously unrecognized problems, providing a context for acknowledged problems.</td>
</tr>
</tbody>
</table>

*Table 1: The potentials of qualitative approaches.*

The studies mentioned above make discernible three particular areas of contribution (see Table 1 above for a summary). First of all, qualitative approaches have been found particularly good at providing rich accounts of practice and thus making it available for inspection, reflection, comment and critique (e.g. Robinson, Segal et al. 2007).
Secondly, unveiling the actuality of software development practice is very important for
the design of tools, techniques or methodologies for supporting practice (e.g. Singer,
Lethbridge et al. 1997).\(^9\) Lastly, the open-ended questions and bottom-up practice
investigations of qualitative approaches are a good way to provide a context for known
problems to be elucidated and the solution space limited (Dittrich, John et al. 2007; see
also Boden, Nett et al. 2007 and Damian, Izquierdo et al. 2007 as examples). This is
also a good way to uncover previously unrecognized problems of software
development. Hence, what researchers performing qualitative research in this area are
doing is not so much testing hypotheses, but describing complexities that are not
obvious and posing questions that can be applied in the improvement of practice or
design of tools, as well as serve as basis for further research.\(^{10}\)

3.3.3 Summary
This section has outlined a stream in research on software development where there is
increased focus on social and situated aspects. This trend also appears in research
specifically on distributed software development (see also Appendix A for more on this
particular topic). Researchers are applying qualitative, interpretive approaches to unveil
such aspects, relying more often on extensive studies of actual, real world software
development activities. By looking at a few specific examples, the potential
contributions of such approaches can be discerned. They allow researchers to elucidate
practice, making the complexities of it available for inspection, providing material that
may inform design of tools, techniques, and methodologies, and posing questions for
further research efforts.

Given the objective of this thesis – to address the issue of how software developers in
distributed projects develop their practical knowledge within particular contexts – the
rich accounts of actual practice that the qualitative approaches discussed here can
provide are particularly appropriate. Through qualitative, interpretive research I will be
able to describe the intricacies and complexities of actual practice, material that is
necessary for providing thick descriptions of how software developer groups deal with

\(^9\) Although the relationship between qualitative research methodologies, particularly ethnography, and
requirements for design is a complex one, e.g. as discussed by Anderson (1994) and Dourish (2006).
\(^{10}\) One important thing to understand when discussing these contributions is that there is no argument
about replacing quantitative, more positivist studies on the subject. Rather, the point is simply that
there are circumstances when it is inappropriate to factor out the complexity that surrounds software
practice (e.g. see Robinson, Segal et al. 2007).
issues of evolving technology, enculturation and socialisation, and emerging project purposes and goals (the topics highlighted in chapter 2).

3.4 Empirical and analytical methods

This section discusses the practical research challenge of studying work practice and outlines particular methods for overcoming it.

3.4.1 The research challenge: unveiling work practice

Studies of work practice seek to dismantle the common-sense conceptions of work, disclose the hidden practices and give us access to the intricate ways and means by which the work is actually accomplished (Schmidt 2000). However, as Suchman (1995) put it, the way in which people work is not always apparent. She argues that assumptions regarding how tasks are performed often are made based on formal analytical perspectives rather than actual underlying work practice. Correspondingly, Orr's (1996) ethnography of a workplace showed how there was a significant difference between espoused, canonical practice and actual practice. The methodological insight here is that exclusively applying quantitative methods in accordance with positivist principles would not be an appropriate way to unveil the complexities of actual practices. Whilst abstractions and formalizations serve a specific purpose, this research endeavour requires methods that can give rich descriptions of what people do and how they do it.

Additionally, it is also necessary to view work practice as a social activity. In presenting their design paradigm the “Workaday World”, Moran and Anderson (1990) were concerned with how people are able to work together the various “co-” processes - communication, collaboration, coordination, cooperation - in teams and other organisational forms. They posited that the basis of this accomplishment is people’s ability to build and maintain social relationships amongst themselves. In other words, the fact that people work together in groups, organisations and communities must be treated as an essential property of working life. This was also highlighted in Orr’s (1996) description of copy machine technicians work practices. While a large part of their work was centred on the technology, an equally important part could be described as the repair and maintenance of the social fabric of the client company. Work practice
Chapter 3 • A qualitative investigation of software developers’ work – methodology and research methods

is, in effect, social practice – with work practice studies we are also challenged to study the social dynamics of the workplace.

3.4.2 Participant observation – the groundwork

In order to investigate and describe the elusive social processes of how people work and use technology in practice, this I have applied participant observation as the main data generation method in two field studies (see chapter 4). It is a method based on ethnographic principles, founded and mainly applied within anthropology and sociology (Hammersley and Atkinson 1996). As we saw in the review of current methodological trends in studies of software developers, participant observation is becoming more common in the domain. The main goal of this method is to gain an understanding of work as seen from the point of view of the participants themselves. This is achieved by spending extended periods of time immersed together with them in their daily working lives. Through a mixture of observations, *in situ* conversations and collection of artefacts, one aims to reveal the activities and interactions of the participants so that one can analyse how their culture “works” (Delamont 2007). “Participation” in this case means, not necessarily to do what the people in the setting are doing (although that can be part of it), but rather *interacting* with them as they are doing it. Through interaction with the people being observed, the researcher can gain insight into what the actors think about what they are doing and how they interpret what is happening, and can also check emerging interpretations. Simply put, the ethnographically-informed data generation technique participant observation is a way for the researcher to gain an intimate familiarity with the dilemmas, frustrations, routines, relationships, and risks that are part of everyday life in software development (Myers 1999). The objective, to produce a thick description (Geertz 1973) of a setting and the actors in it.

Whilst collecting documents, taking photographs, and, where possible, making video and audio recordings are excellent ways of gathering data, it is extensive note taking on behalf of the researcher that forms the main method of record in this type of study. Of course, the key issue is to know what to observe, how to observe and, in particular, what to write down (Delamont 2007). A prior literature review is essential to let the researcher define the purpose of the study and what set of issues are to be investigated. But an important ethnographic principle is to let the experiences in the field (or the
“data”) lead to a full understanding of the culture being studied (Hammersley and Atkinson 1996). Therefore, the researcher, to her best ability, observes everything she can and writes the most detailed field notes possible. It is important to record the naturally occurring talks and interactions among people, and this is best done through ongoing field notes (Pentimalli 2000). But it is equally important to take time to reflect, elaborate, and expand on them outside the field or as soon as time permits. This is usually done by keeping a fieldwork diary (ibid.). The key with this is that the method of record must not be seen as just a way of simply gathering “data” – participant observation is a process (Delamont 2007). The research proceeds in a series of loops and in each step the researcher reflect upon, and even revisit, earlier steps. The taking of notes and writing up the material are vital parts of this process. In the words of Delamont: “Essentially an ethnographer observes everything she can, writes the most detailed field notes she can, takes time to expand, elaborate and reflect on them outside the field and/or as soon as time permits, constantly pesters those being observed to explain what they are doing and why, and sweeps up any documents, pictures or ephemera available” (ibid. p. 213).

As pointed out by Myers (1999), one must also be aware that this method of research does have areas open to critique, just like any research method. First of all, it requires an extensive amount of time. Not only does the fieldwork itself take time, but the process of analysing the material is equally, if not more, time consuming (I discuss my analytical methods in the following section). This type of research is perhaps most suitable where such an amount time is available, e.g. during doctoral studies. However, it must also be pointed out that participant observation, if done thoroughly, is still productive in the sense of amount and substance of the findings. Secondly, a case study based on participant observation leads to in-depth knowledge of only one particular case. Critics claim that there is a problem of generalizability. However, it may also be argued that this is partly due to the novelty of this approach in the particular area of technology in organisations (ibid.) – as more studies get done, it will be possible to develop more general models of the meaningful contexts being studied. Furthermore, any type of case study has this problem (Yin 2003), and there are also methods for generalizing results from case studies which are quite successful (Myers 1999).

Myers also discusses various criteria on which to evaluate ethnographic studies (Myers
1999). First of all, two criteria relate to the value of the findings: to what extent does the researcher contribute something new to the field, and are the rich insights into the subject matter valid? Considering these aspects is naturally important for judging the value of any type of research endeavour and it is achieved mainly by situating the study in a wider body of research for comparison. But because of the nature of the qualitative method applied, another two criteria are critical for ensuring validity: has a significant amount of material been collected, and is there sufficient information about the research method? Describing in detail the actual steps of the research, how it evolved over time, and the assumptions that it was based on is very important because of the ideographic nature of participant observation. It is based on this that principles for evaluating interpretive case studies can be applied (Klein and Myers 1999). This also brings attention back to the importance of reflexivity in this type of research, as we saw pointed out by Delamont above (Delamont 2007). Reflection is an essential part of the process itself, but it's also apparent that reflection of the process through which the ethnography grew is essential in the presentation of the results.

3.4.3 Grounded theory coding – coping with the flood

The empirical material produced during participant observation is very detailed and, with hundreds of pages with notes and field work diaries, vast in scale. This represents a significant challenge for how the researcher is to draw out and analyse the content. In the research presented in this thesis, principles and techniques from grounded theory (Glaser and Strauss 1967; Strauss and Corbin 1990) have been applied to cope with this challenge. Grounded theory is today mainly associated with a set of procedures and techniques for analysing data (Dey 2007). But in its original conception, it is an entire strategy for qualitative research studies (Glaser and Strauss 1967), including basic methodological assumptions, processes for sampling and selecting cases and sources, and a set of empirical methods and techniques for analysing data. The research presented here has not adopted a full grounded theory strategy as its methodology. But the empirically grounded approach of grounded theory has informed the field study processes and the coding techniques have been used as a way of analysing the empirical material.

The key idea of grounded theory is that theory should be generated through research
data (Glaser and Strauss 1967). That is, rather than testing preconceived ideas, researchers discover new theoretical insights by engaging with actual evidence from “the real world”. There is also a strong practical orientation, as theories are judged on their relevance and utility (Dey 2007). The principal of empirically grounding the research effort has guided the design of the studies on which this thesis build. They've adopted an open-ended approach, characterized by exploratory investigation and dialogue between posed questions and empirical findings. Some setting and certain activities have been studied and analysed, iteratively leading to more developed research questions and further investigation of certain aspects. In terms of results, this meant that it was not possible to know from the outset what the results would tell (it was not about testing hypotheses), only what aspects or issues were of interest. But since the purpose of the studies was exploratory and descriptive, and since strong links to observational evidence was necessary for unveiling the actual work practices of the settings, it was an appropriate approach for this thesis research.

A grounded theory coding process (Strauss and Corbin 1990) has also been adopted for analysing the empirical material that was produced through participant observation. It provided a natural way of breaking down the vast material and assessing the aspects important for the investigation. The coding process itself is essentially about selecting pieces of data and sorting them under categories. As the material is worked up this way, significant categories begin to emerge. Strauss and Corbin (ibid.) distinguish three phases through which this happens. Open coding is the initial analytical effort. The researcher begins breaking down the data which is generated and creates categories which conceptualise what has been observed. Axial coding is the next phase, wherein the researcher begins making connections between the emerging categories, thus forming a more integrated understanding of the data. Lastly, through selective coding, the researcher identifies a core category around which the others can be organized as support. This technique of selecting and categorising data was applied both during the field studies and in the retrospective analysis of the material when it was being written up into a thesis. During the field studies, coding was the systematic method which guided the inquiry. After the field studies were completed, more detailed analysis ensued, and line by line coding of the produced material was the main method. I adhered to Strauss and Corbins (1990) principle of referring to literature when
necessary during this process in order to situate my results within the boundaries of the study area. More detail on the actual process and the outcomes will be given with the field study descriptions of chapter 4 (particularly sections 4.2.2 and 4.3.2) and in the discussion of selection and representativeness of vignettes in chapter 5 (section 5.2).

3.4.4 Field studies and thesis layout

Using the empirical and analytical methods discussed in the previous section, I have performed two field studies for the purpose of this thesis. The two cases represent two different settings of distributed software development: a multinational organisation and an open source community. Case studies were appropriate because the issues in focus need to be investigated in real-life contexts (Yin 2003). The reason for studying these two particular cases was not to evaluate the differences between proprietary software development in organisations and open source software development in volunteer-based communities. Instead, the argument is that corporations and open source communities essentially are different settings for the same substantial activity. This selection of cases allows me to show dealing with changing circumstances within distributed software development practice in different settings. The difference in setting also give rise to contrasts that are useful in the analysis of what characterizes this problem (the relationship between the cases is further discussed in chapter 4).

The strategy for writing up the material produced on these two field studies is descriptive (Yin 2003), aiming to provide a descriptive framework of each case. These will be presented in detail in chapter 4, including detailed accounts of the study processes and findings in form of analytical categories that emerged through grounded theory coding. The purpose is to provide an initial thick description of work practices in both settings, containing the first set of evidence for my argumentation. Following this, four episodes of practice – two from each case – are extracted and rendered as vignettes in chapter 5. Each illustrate in detail an episode of changing circumstances, exemplifying the topics suggested in chapter 2 (introduced in section 2.5.2). This chapter provides an explanation building analysis (Yin 2003) of the episodes, tracing out the chain of events chronologically and attempting to explain them by stipulating a presumed set of links between activities and events. This helps produce insights into how issues of evolving technology, enculturation and socialisation, and emerging
Chapter 3 • A qualitative investigation of software developers work – methodology and research methods

Having presented these empirical analyses, the next phase of the thesis is interpreting the findings and addressing the research question. In chapter 6, I identify and discuss the common themes that emerged across the different episodes. This serves as an important step towards the main objective of this thesis by characterising how change of circumstances unfolds and thus what situations that the software developers involved must deal with. In chapter 7, attention is shifted back towards the main issue being addressed with the thesis, namely practical knowledge. The discussion will focus on how the software developer groups responded to issues of change as seen in evidence amassed throughout the preceding empirical analyses. Using the insights from the discussion of the themes that characterise how changing circumstances unfold, the main concerns of the software developers are explained. This is then re-interpreted from a practice-based perspective as practical knowledge being developed in response to constantly changing circumstances, hence addressing the research question.

Thus, the analytical approach for addressing the research question in this thesis is narrative in style, in the sense described by Becker (1998 p. 57-58 & 60-63). Focus is on the process by which something happened or came to be. Rather than showing why something was or became necessary, it is showing how something was or became possible. By focusing on how episodic events unfolded in real-life contexts, I can describe the important dynamics of how practical knowledge is developed as change in circumstances occur and is dealt with.

3.4.5 Summary

In this section, the research challenge was described as one of unveiling actual work practices. Participant observation and grounded theory coding techniques have been the methods utilized for addressing this challenge and producing empirical material in two field studies. In this thesis, the material is presented first in accounts of the two cases, and then in form of vignettes that illustrate the three topics suggested in chapter 2 as manifested in the particular contexts under study. These thick descriptions are then discussed, first in terms of what the common themes between the different episodes are, and then in terms of how the software developers dealt with the situations. This is a narrative style of analysis.
3.5 Chapter summary and conclusions

In this chapter, I have presented the methodological framework of my research. The research effort was part of the socGSD project and the methodological background is related to ethnographically-informed empirical studies of software engineering, and work practice studies in the computer-supported collaborative work and organisation science disciplines. The approach is qualitative and interpretive, aligning with current a stream in the software development research domain.

Participant observation and grounded theory coding techniques have been utilized to generate and analyse empirical material on work practice in two different settings of distributed software development. In the thesis itself, the style of analysis is narrative, as the material is written up as thick descriptions. Following a detailed account of the findings from the two field studies (chapter 4), four episodes of practice are extracted and rendered as vignettes (chapter 5). They illustrate issues of evolving technology, enculturation and socialisation, and emerging project purposes and goals (introduced in section 2.5.2) within the particular contexts under study.

The thesis then proceeds in two stages to interpret the findings and address the research question. First, the shared themes between the different episodes are identified and described in general terms as the problem of changing circumstances (chapter 6). Then, focus is shifted towards how the software developers dealt with this problem, the argument being that this happens through the purposeful adaptation of practice (chapter 7). This is re-interpreted from a practice-based perspective as the development of practical knowledge.

The methodology and research methods outlined in this chapter informed the two field studies which are included in this thesis. Having thus set up a backdrop to help the reader understand particular methodological choices, the following chapter presents the two studies in detail.
4 Elucidating distributed software development practice – distribution as a mundane issue

4.1 Introduction

In this chapter, I elucidate work practice in two different settings of distributed software development. I do this by presenting in detail two field studies that have been conducted for this thesis. These have been performed within the research context and methodological framework outlined in chapter 3. The material produced is ethnographic in nature and the results are a number of analytical categories that conceptualize work practice in the two settings.

Within the grander scheme of this thesis, the purpose of this chapter is to provide a first set of evidence about the dynamics of work in distributed software development settings and for my argument about how software developers in distributed projects develop their practical knowledge while engaging in practice (see chapter 7). But, importantly, it also provides an empirical context within which the particular issue of changing circumstances can be further explored (see chapters 5 and 6).

The first study presented in this chapter is a observational field study of a development team of a globally distributed project in a multinational organisation. I have given the team the pseudonym Lambda. Observations of the Lambda team scoped 29 full working days within 5 months, and the purpose was to investigate the work practices of a local team in a distributed environment. Grounded theory coding techniques were used to analyse the empirical material as it was generated through participant observation.
The second study is an exploratory case study of an open source community called PyPy. Focus was on investigating the practices of sprint-driven development in this particular community. The study was ongoing for an extensive period of time - roughly 2 years in total - and underwent many different phases as the case was explored. The methods utilized range from participant observation, *in situ* interviews, and an ethnographic experience, to qualitative analysis of online discussion forums\(^\text{11}\) and open-ended email questionnaires. Grounded theory coding has been the method for analysing of the conglomerate of material so produced.

### 4.2 The Lambda field study – the workaday in a distributed project

The first field study we are going to look at took place within a multinational corporation, where I was allowed to observe the work of a team working in a globally distributed project. I will give the project the pseudonym “Lambda” and refer to the team I observed as the “Lambda team”. The Lambda team were the development team of the globally distributed Lambda project, meaning that they collaborated with test teams and other partners from several of the organisations sites around the world, including China, Germany, and the USA. The Lambda team themselves were based in the company's Dublin office, which was the site of study. Thus, this case represents distributed software development in a proprietary, organisational setting. Distribution came, not from outsourcing or inter-organisational collaboration, but from the fact that it was a global organisation that could utilize competences from different parts of the world. As we shall see, this was something that simply was taken for granted by the Lambda team members. Distribution was an inherent part of the work, not something new or exotic, but simply something to cope with.

The Lambda team became one of my cases by a combination of choice and opportunity. Global organisations are, alongside outsourcing relationships and open source communities, one of the natural places for distributed software development. As such, it represents a primary setting for research on the subject, above all offering opportunities to observe structured and organized ways of achieving the task. However, gaining

---

\(^{11}\) It should be noted that PyPy's online forums – including both email list archives and IRC chat archives – are publicly available via their webpage at [http://codespeak.net/pypy/](http://codespeak.net/pypy/) (Accessed: 2010-02-05). For the interested reader, I have also included links throughout the text where I extract quotes from these forums (both in this chapter and in chapter 5).
access to corporations can be difficult. I was searching for an appropriate case for a few months, getting in contact with various multinational companies that had software development sites in Ireland. But it was two socGSD project members relationship with the Lambda team's organisation that finally gave me a real opportunity. My colleagues were performing studies in the organisation at the time, and had been doing so for roughly a year. Having spent much time there, they had built up a close relationship with many people and had good insight into project developments on the site. As it happened, the project manager and product architect on the team they observed were reassigned to a new project on the site during the spring of 2007. The manager and architect became the leaders of this new project and, since they had worked with academics for some time, were open to the idea of having researchers participate. After my colleagues made the introductions, I was offered the opportunity to perform field studies with the new project. Having negotiated access with the managers and discussed my intentions with the team members, the study began in late August 2007.

I undertook an ethnographically-informed field study, aimed at investigating the issues of distributed software development from the point of view of the Lambda team members. Instead of trying to explain this set of issues in terms of organisational procedures or software engineering processes, the study centred on people and what they were doing to accomplish the job. This meant taking a local perspective on distributed software development. I was not looking at the Lambda project as a whole, trying to explain the collaborative dynamics between the sites, but was studying distributed software development in terms of what it meant in practice. That is, the study aimed to investigate how software development was achieved in practice in the context of a distributed project.

4.2.1 The Lambda project

4.2.1.1 Project background and objectives

The team in focus – which have been given the pseudonym “the Lambda team” – were based in the Dublin office of a multinational organisation. They were the development team of the globally distributed Lambda project. As such, they constituted the core of the project, which thus was centred in the Dublin office (see section 4.2.1.2 for more on the project architecture). The Lambda project went into its second phase after the
release of the 1.0 version of the product in June 2007. Continued development was ultimately aimed at the next generation of the product (referred to as “product 2.0” from here on), but also included supporting and incrementally improving the already released version (we'll say “1.x patches”).

When I began my observations in August 2007, it was a volatile time for the Lambda team, as there was uncertainty regarding the direction of the project and how much resources the organisation would devote to it. Following the 1.0 release, the Lambda team had roughly 30 members and the initial intention was for it to grow, potentially all the way up to 70. New people were assigned and recruited. However, when I began participating, there had been a drastic change of plan, as the organisation had recently decided to put significantly less resources into the project. The number of members in the Lambda team dropped from an expected 30 and above, to around 15-16. As a consequence, there was also a change in leadership, as the HR manager was transferred to another project. The development manager and project manager continued on, but the latter was also reassigned only a month later. He was replaced by a junior manager, for whom this was the first experience of independently managing the day-to-day organising of a project. Even though the uncertainty continued to hamper the team for the duration of my study (e.g. discussed in section 4.2.3.5), the membership roster finally settled on 17 people - including developers, technical leaders, managers, and the product architect – and this remained largely unchanged until the end of my study in January 2008.

The product developed in the Lambda project constituted a set of portlets for an enterprise portal software framework. It utilised information from a well-known enterprise resource planning (ERP) system and provided self-service human resource functionality for both managers and employees through the portal framework. It was thus an integral part of a wider software framework, as it provided an interface between the ERP and the enterprise portal framework. Apart from mastering software development methods and programming in general, development also required knowledge about the functionality of these two complex software structures, as well as numerous other software techniques necessary for implementing the product (e.g. various server, security and access control schemes and protocols). There was a high dependency on these external software products and the team had to cope with constant
changes in these as they were independently evolving.

For development process, the Lambda project employed certain principles from the Scrum agile method. They had an incremental development process, with 5 to 6 weeks long iterations – development sprints. The goal of each was to achieve major product milestones, but exactly what that meant was only decided at the start of each sprint, presumably allowing for an agile process. This decision was in turn based on what was actually achieved in the preceding sprint, which was evaluated in wrap-up and lessons learned meetings. While the wrap-up meetings were more formal affairs aimed at reporting what was achieved, the lessons learned meetings were meant as forums where continuous process improvements could be worked out.

4.2.1.2 Sites and relationships
This section details the overall architecture of the Lambda project. For an overview of the sites and relationships of the project, see Figure 2 on page 56. Product and release management of the Lambda project was handled from the US and there were also three test teams involved in the product development, one located in the Dublin office and two in China. Collaboration between these partners was regular, both on a management level and on a day-to-day work level. In addition, a server running the ERP system which the product utilized had been set up as part of the development environment. However, this server was run and maintained at a site in Germany, which added another interrelation to a geographically distant team. Lastly, since the 1.0 version of the product was on the market, there was a need for customer support. Support was primarily handled by a team in the US. They worked mainly with technical and functional documentation provided by the Lambda development team. But complex and problematic issues were communicated to the development team for solutions. The support team also reported any fallacies and inconsistencies in the documentation, which had to be maintained by the Lambda team.

4.2.2 Research method – an observational field study
I followed the Lambda team over a period of roughly 5 months, from late August 2007 to the end of January 2008. During that time I was allowed to occupy a desk in the open office area where the team resided and to shadow the team as they were working. I was free to interact with anyone to such an extent that it did not interfere with their work. As
well as naturally occurring casual conversions, open interviews in relation to the various situations and events I experienced with the team were numerous and an important element of the study. I was also allowed to attend all the meetings of the project. The Lambda teams meetings primarily included weekly team meetings and technical leader meetings, but project-wide meetings – facilitated by a telephone conference system - were also an important segment of the teams activities that I followed closely. Additionally, I subscribed to the projects frequently used internal mailing lists and had access to much of the project documentation (not including source code).
In total, I spent 29 full working days with the Lambda team, excluding early visits, for example when negotiating access and making introductions to the team. This time included both periods of relative calm and periods of intensive activity. For example, I was present during one version update release and several milestone wrap-ups. The main method of record was note taking, including ongoing field notes, specific notes at meetings and interviews, and a fieldwork diary written at the end of each weekly visit (on average 2-3 days long). Apart from this I also collected a rather large amount of electronic material, for instance in the form of conversations from the mailing lists, articles from the internal Wiki, and shared documents, plans, and discussions from the intranet pages.

My observational effort to study the Lambda team was exploratory in nature, focused on work practices in the setting and attempting to identify interesting lines of inquiry in relation to these. The technique I was applying to identify relevant lines of inquiry roughly corresponded to what Strauss and Corbin (1990) terms “open coding”, in that I would label and group observations. As the study carried on, certain chains of events thus emerged as “stories” that played a significant role in the teams working experience. As a response, my continued observations would focus more intently on following these stories. The principle of theoretical saturation was then used as a cue for when the field studies should be concluded. That is, once further data no longer prompted new distinctions or refinements in the propositions about the case, it was judged that there was enough empirical evidence. In the end, what this helped me produce were very detailed accounts of specific situations and practices in the Lambda team.

The coding performed during the field studies helped me narrow down my observations to significant aspects of the Lambda teams workaday. Its main purpose was to identify these aspects and guide the empirical investigation towards gathering material about just those aspects. After the field studies were completed, however, there was time for more detailed analysis of the generated empirical material. With the help of the NVivo 8 software suite, the material was re-coded line-by-line in several iterations. This allowed me to analyse in more detail the stories highlighted during the fieldwork, especially in terms of what characteristics they had in common. They were individual

12 The fieldwork diary contained a general account of the visit, a list of people interacted with, a summary of the main insights and ideas sparked, and finally a rough plan for the next visit.
chains of events, but now it could be seen how they were interconnected and related.

The coding procedure was iterative and included open, axial, and selective coding (Strauss and Corbin 1990). The entire collection of empirical material (field notes, fieldwork diaries, and collected artefacts) was first conceptualized, resulting in a large number of codes (more than 50). In a second iteration, I went through the body of material highlighted through the first round again, re-using the existing codes, but also combining and re-labelling them according to perceived relationships and alternative interpretations (discarding codes not containing sufficient evidence for thick descriptions). The outcome was a smaller number of more refined codes (14). By now I had begun working manually on a whiteboard in parallel with using Nvivo 8, because I felt that this gave me a better overview. Figure 3 below contains an image of the codes during the final stage of the second iteration. The post-it notes are the refined codes and the handwritten tree structures the original codes. Lastly, I proceeded by grouping the various codes that had now emerged, effectively creating a number of analytical categories under which the major codes could be sorted. The categories were created from major codes – codes that contained a significant amount of empirical material and related to a large number of other codes – either by simply expanding them into categories or by grouping them under a new overarching categorical label. From this
coding process emerged five significant analytical categories (that is, groups of major
codes under a common descriptive label), conceptualising the work performed by the
Lambda team. These five categories are the core of my analysis of the Lambda case and
will be presented in detail in the following section.

4.2.3 Concerns of a software development team in a distributed project
In this sub-section, we shall look at the significant analytical categories that emerged in
my study of the Lambda team. They are the outcomes of the grounded theory coding
process (see the previous section) and represent the main findings about what
characterised the workaday in this team.

4.2.3.1 Solving problems together
When discussing what defined the workaday in the Lambda team, a most obvious – yet
one easily forgotten – theme regards what the actual job was about. The main
engagement, of course, was software development activities. Essentially, this can be
described as solving problems, for example in terms of how to implement a certain
feature or functionality, or how to fix a reported bug in the code. But, interestingly, what
emerged as a significant analytical category in my study were the collaborative aspects
of this. What was most frequently seen during the day to day work was people
discussing how to solve these problems, explaining them to each other, proposing and
debating different solutions. Because software is an abstract construct, this can be seen
as a constant negotiation to agree on the structure - e.g. the relationship between
components, where certain things are processed, who is responsible for what, and so on
- of the product. Important for these discussions was also a shared knowledge about
common solutions, structures, and norms of the software development domain, as this
enabled discussing specific solutions to the problem at hand.

Furthermore, a recurring observation was how often these discussions would boil down
to talk about who else might be working on similar or related issues, within the team,
but just as often in the overall site or organisation. The majority of tasks would in fact
be in some way be related to other projects or systems, e.g. it could be writing bridges
and interfaces between the Lambda product and other systems, or developing
installation configurations for the product to be installed on a variety on platforms, and
so on. Knowing what others were doing was therefore a very important part of the job.
Interaction was also frequent when it came to supporting activities within the Lambda team. For example, setting up and maintaining the intricate development environment was a complex task, but having all the right processes and frameworks in place was a must for development. People assisting each other in using or configuring the development tools of the site was a common feature of the daily work. Another common support activity was keeping track of what was expected from the participants at meetings and this would often take the form of people helping each other prepare for the events.

The theme here, then, is collaborative problem solving, and in it we see elements of both articulation work and communities of practice. The cooperative work of the Lambda team can be viewed as individual yet interdependent activities that had to be “coordinated, scheduled, aligned, meshed, integrated, etc [...]” (Schmidt and Simone 1996). It required the participants to engage in activities extraneous to the activities that directly contributed to the object of work, which is what the concept of articulation work refers to (Schmidt and Bannon 1992). Additionally, the fact that people thought about their work in terms of relationships extending well outside the team itself can be explained with the view that they worked in communities of practices that transcended organisational boundaries (Wenger 1998). In the context of a distributed project, this also contributes to an insight that work in such settings is not – on a fundamental level - distinctly different from work in organisations as described from practice-based perspectives.

4.2.3.2 Coping with remote dependencies

In section 4.2.1.2, I presented the relationships between sites in the Lambda project. Even though the Lambda team were the developers of the product, the development work could not be done without coordination and collaboration between all the various teams and partners involved in the project. Consider, for example, the testing process. Somewhat simplified, it was a matter of each team looking at a specific aspect of the system. The Lambda development team verified that the product built properly. That is, that the code worked. The Dublin test team looked at it from an emergent system perspective, e.g. testing how the product functioned over time on a variety of systems. One of the Chinese test teams focused on performance testing, while the other tested the
actual functionality of the product. In addition to this, the US support team would report fallacies and inconsistencies with the technical and functional documentation.

To a large extent this all happened in parallel. Defects and change requests reported from the various tests would continuously be added to the Lambda teams task lists. It often required developers to communicate with the test teams, for example to ask for more information on a reported bug. Additionally, when a test team had reported a bug or a change request, they also had to approve before it was signed off as fixed or deferred. This made detailed coordination necessary and it was one of the main tasks for the managers of the project. There were weekly conference call meetings where the managers and leaders from each team reported to each other and discussed how the work was to progress.

Concerns related to this distributed dimension of the project did emerge as an analytical category in my study of the Lambda teams daily work. To being with, dependencies on remote teams, especially for testing, often meant a staggered work flow for the Lambda team. They frequently found themselves waiting for reports on the latest builds before they could proceed. This is nothing unusual in large-scale software development, but there were observations which suggest that distance had a slight impact on this collaboration. Particularly, I perceived that distance might have impeded the partners reaching a common understanding of work items. Coordination between the remote sites was mainly achieved in the weekly conference call meetings where leaders of the different teams of the project would report and confer. This meant that direct interaction between the teams was rather infrequent and that misunderstandings could linger longer than necessary, although such problems could often be alleviated by informal communication, e.g. via email conversations. At the conference call meetings themselves, much effort went into explaining what exactly was referred to with specific work items. There were also problems due to different practices and environments. For example, of bugs that were deferred because they could not be reproduced or explained outside of the specific site environment where they occurred. However, my main observation regarding these issues is that they did not take any primary role in the daily concerns of the Lambda team, as they were not discussed outside the specific meetings or particular incidents.

For the developers of the Lambda team, the distributed dimension of the project also
played a small role. The most frequent activity related to it would be when working on reported bugs or change requests and there was a need to send remote colleagues an email to ask for more information about it. There were more noticeable activities as well, for example when the developer responsible for the products installation procedures was assisting the US customer support team with setting up their systems. But when asked about it, he described it as non-problematic. He used common tools for remote interaction – screen sharing and VOIP\textsuperscript{14} facilities that were part of the organisations information technology infrastructure – and did not view it as anything out of the ordinary. Another example was the ERP systems expert of the project. She was part of the Lambda development team, but her task was to support both local and remote colleagues with issues related to the ERP system. She described to me how she noticed a difference in how she was approached by local colleagues compared to remote colleagues. Whereas the local people would simply ask for help and accept the solution given, people from other sites – particularly the Chinese testers – would often insist on her explaining \textit{why} the solution worked. This could be explained as a matter of the remote partners feeling it was cumbersome to contact her and thus wanted to learn how to solve more problems themselves in the future, whereas the people in Dublin simply assumed she would be available for future questions since she was sitting just a few desks away in the open office area. But yet again, my observation is that examples such as these took a back-seat role in the day-to-day work.

As I wanted to investigate further the Lambda team members perception of issues relating to distribution, I began asking team leaders and developers about what it was like to work in a “globally distributed project”. However, I was met mostly with slight bewilderment. In their perception, they were members of the Lambda team, a coherent group located in the same open office space. The collaboration with remote partners was an inherent and vital part of the work. Yet it was taken for granted by the people working on the team. For them, it was just the way things were. Coping with it was part of the job. My conclusion is that, while there were clearly activities related to dealing with the dependencies to remote partners, these did not play a outstanding role in the work of the Lambda team. Distribution was a mundane issue, one of the many things

\textsuperscript{14} “Voice Over Internet Protocol” is a term for the family of technologies that transfer voice communication over IP (Internet Protocol) networks, such as the Internet and other packet-switched networks.
that it was their job to deal with.

### 4.2.3.3 Adjusting practices

Another analytical category that emerged in my study of the Lambda team relates to the organising of team activities. The reassignment and recruitment period that followed the 1.0 product release (section 4.2.1.1 contains a brief history) meant that the Lambda team mostly consisted of people who had not worked together before. Many were even new to the organisation, and this was combined with recent leadership changes as well. Therefore, neither the processes nor the actual practices for team activities were a given and the team – as will be shown below - spent a lot of time figuring out how to do things together, often by trial and error.

One example was the weekly team meetings. They were obliged to have them and it was intended as a way of keeping everyone informed about the teams overall status and activities, but nobody had a clear idea of exactly how to organise those meetings. In the beginning of my time with the team, they experimented with round-table reports from everyone on the team. But this was found cumbersome and not very informative. After that foray they switched to having component leads give a brief overview of developments, followed by a demo from one or two developers who had achieved something concrete since the last meeting.

Another example was the wrap-up and lessons learned meetings that concluded each development sprint. The project manager was junior to the organisation (again, refer to section 4.2.1.1 for a detailed context), so she had no experience of organising and leading such meetings. No one else had any detailed knowledge about Scrum either. (Interestingly, her predecessor had been an expert on the Scrum method and the one who championed them within the Lambda team – she effectively inherited the requirements set in place by him). Therefore, the wrap-up and lessons learned meetings were themselves a learning process for the team. During the time of my study there were two consecutive development sprints concluded and I observed significant changes between the two. Firstly, whereas the first one was organised as a single meeting for the whole team, the second consisted of each sub-group of the team having their own meetings. Having consulted both fellow team leaders and developers, the project manager had come to realize that the presence of team leaders and managers impeded
discussions somewhat. Therefore, team leaders did not participate in their respective groups at the next development sprint wrap-up, but instead had their own meetings, which also included reflecting on the feedback from the preceding group meetings. Secondly, having also become aware of concerns about the balance between focus on negative and positive aspects of the work at the first meeting that I observed, the project manager also attempted to steer subject matter more in the second. While still an open agenda, the goal then was to discuss in order, what went well, what didn't go well, and what could be done better in the next sprint.

What this analytical category depicts, is how the Lambda team were gradually adjusting their practices. This is an important insight for my argumentation about how software developers in distributed projects develop their practical knowledge through the purposeful adaptation of practice, and I will draw on it further in chapter 7.

4.2.3.4 Going off track – with a purpose!

Interestingly, amidst the activities described so far, there were more avocational activities that also emerged as a significant analytical category. During my time with the team, one of the main stories (see section 4.2.2 ) regarded an event that allowed developers to take a full working day to work on whatever they wanted. It was followed up by a show-and-tell seminar where people presented and discussed each others achievements. The event was organized worldwide within the organisation - although the particular details were organized locally – and developers were especially encouraged to access expertise and code from all over the organisation in pursuit of fulfilling their own ideas.

What was really interesting was how keen the Lambda team leaders – including the top managers on site – were for people to participate. Numerous encouraging emails about the event were sent to the mailing lists and brainstorming meetings were organized within the team in preparation for the day. These meetings were meant to allow discussion of ideas, and to provide inspiration from others and encourage potential collaboration. Ideas related to areas that the developers wouldn't normally work on were prioritized. When asked about the reasons for this propagation, one of the team leaders explained to me that it was a chance for the developers to "think outside the box" and do stuff they thought was fun and interesting, as well as getting to work with people in
other teams.

Despite almost complete participation in the brainstorm meetings, and much discussion of ideas at that, only one developer from the Lambda team participated in the event. After the event had taken place, I interviewed a developer on the Lambda team who had participated in it. I was particularly interested in his own reasons for participating, what he actually ended up doing, how he went about it and who he worked with. The first thing I was interested in was why he thought it was worthwhile to spend time on the event, working on something that wasn't directly related to his daily tasks. His initial response was: “it's a chance to work on something else.” Considering the matter for a moment, he added learning new technology as a key thing, but also went on to portray it as a recreational activity. Importantly, he also mentioned the chance to get in contact with people from other projects. Indeed, his achievements on the day were noticed by a team in another office of the organisation, who contacted him to discuss the possibilities of including his work in their project.

There were other - albeit none as encompassing or high-profile - events of a similar nature as well. For example, there was a lunch time seminar series on site which developers could, and often would, attend. It covered a variety of technological topics, e.g. like setting up a Linux server. In relation to these events, a recurring topic of discussion among team leaders was how to ensure that developers had the right knowledge and skill for handling their tasks. Additionally, developers were often encouraged to develop their skills through training courses. For example, at one point, the fact that there were places available in an advanced Java certificate course on site was advertised to the team by the development manager.

The activities contained in this analytical category can in some ways be described as going off the track, in the sense that they were not directly related to the work task of the team. But they were in fact highly valued and much encouraged, primarily because it was a way to both develop skills and to expand ones personal network within, and knowledge about, the global organisation which surrounded the team. The “global” must be emphasised here, since the collaboration in what I have described above extended well outside the Dublin office where the Lambda team were based. Also, there are important insights here about the importance of professional and technological relationships. The Lambda team were not isolated from the broader world of software
development, something that I will return to in chapter 6 when I characterise the problem of changing circumstances (see especially section 6.4.1).

4.2.3.5 Dealing with uncertainties and change

Another significant analytical category that emerged through the study of the Lambda teams work practices, was how they were dealing with constant uncertainties regarding the project goals and continual change of circumstances. When I joined the team, the project goals were being re-negotiated. The delivery date for the 2.0 product was extended by almost six months and decisions had to be made regarding what functionality could actually be delivered. But this uncertainty remained throughout my time of study. This was clearly illustrated at a project meeting in January 2008. The main topic of discussion was the uncertainty regarding what they were actually working towards, and there were no less than 6 scenarios for what they would deliver in the end. The fact that they were working with an evolving plan also related to the agile development methods they applied. There was no specified project plan, but rather goals were set for the 6 week development sprints. This was a way of dealing with the fact that while there were no long-term plans, development still had to proceed.

In addition to this, the Lambda team also had to deal with a series change of circumstances. The organisational changes following the 1.0 release (described in section 4.2.1) required adaptation in terms of responsibilities and readjustments of what was achievable within one sprint. As responsibilities changed, the availability of people for various unforeseen tasks became a constant concern. Furthermore, since the product under development was an intricate part of a wider software framework, the team had to deal with the autonomous development of framework systems. There were also changes in the technological infrastructure available to the team, specifically as the collaborative platform was replaced within the project (an episode that I specifically explore in a vignette in chapter 5).

Issues of uncertainty and change were thus salient in the Lambda teams work practices. This is a particularly important insight in the context of this thesis, since its overarching goal is to analyse how change of circumstances are dealt with in practice. The following chapter (number 5) will explore this aspect in more detail, particularly in a vignette that illustrates the changes in technological infrastructure within the project.
4.2.4 Summary and discussion

In this section we have seen the results of 5 months participant observation with a software development team in a multinational organisation. They were part of a globally distributed project and I was allowed to join the team during their daily work in the organisations Dublin office. The purpose was to examine distributed software development from a local perspective, in terms of the work practices of a team in a distributed project. Through a grounded theory coding process, five analytical categories emerged that characterise the work in this setting. I summarise the findings in text below, but see also Table 2 on page 68 for a condensed summary.

First of all, collaborative problem solving that transcended organisational boundaries showed us how work practice in this distributed setting not was fundamentally different from the image of work in organisations as commonly described by scholars. How the team gradually were adjusting their practices for team activities was another analytical category, and it is important for my discussion of the purposeful adaptation of practice in chapter 7. Activities that effectively meant going of track – in terms of not relating directly to the teams work tasks – also emerged as important, particularly as a way to develop skills and expand ones personal network within the global organisation. A fourth analytical category regarded dealing with the distributed dimension of the project, but, interestingly, this was found to be a largely unproblematic and wholly mundane issue. Lastly, the fifth analytical category that emerged related to dealing with uncertainty and changing circumstances, an aspect that will be further explored in the following chapters.

The insights of these findings do two things within the context of this thesis. First, we get an image of distribution as a mundane issue within an organisational setting, as something that simply is a casual part of the workaday. Secondly, they unearth evidence for the prominence of continuously changing circumstances and gradual adjustment of practice, something that is important for the overarching argument of this thesis. I will explore this aspect in more detail in the following chapters (5 and 6) and will also draw on the material presented here in chapter 7.
<table>
<thead>
<tr>
<th>Analytical category</th>
<th>Findings</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collaborative problem solving</td>
<td>· Elements of articulation work and communities of practice in daily work</td>
<td>Work practice in the team corresponded to the image produced by practice-based perspectives.</td>
</tr>
<tr>
<td>Coping with remote dependencies</td>
<td>· Coordination among team leaders through weekly meetings</td>
<td>Distribution was a mundane issue, something to deal with as part of the daily work.</td>
</tr>
<tr>
<td></td>
<td>· Occasional informal communication and extra activities in everyday development work</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· Did not see themselves as working in a “globally distributed project”</td>
<td></td>
</tr>
<tr>
<td>Adjusting practices</td>
<td>· Team were gradually working out what to do at scheduled team activities, such as weekly team meetings and lessons learned meetings</td>
<td>The team was gradually learning how to do things together.</td>
</tr>
<tr>
<td>Going of track – with a purpose!</td>
<td>· Developers were encouraged to, and did, participate in worldwide event within the organisation for spending a day working on whatever topic they wanted</td>
<td>A way to develop software development skills, but also to expand personal network within, and knowledge about, the global organisation.</td>
</tr>
<tr>
<td></td>
<td>· Activities such as training courses and technical seminars were common elements</td>
<td></td>
</tr>
<tr>
<td>Dealing with uncertainties and change</td>
<td>· Constant uncertainties regarding what the project goals were</td>
<td>Issues of uncertainty and change were a salient part of the teams everyday work.</td>
</tr>
<tr>
<td></td>
<td>· A series of changing circumstances in terms of team organisation and technological infrastructure</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Summary of findings from Lambda field study.
In turning to the second study that is included in this thesis, we're shifting attention from a proprietary, organisational setting to an *ad hoc*, volunteer-based one, namely the PyPy open source community. They are employing a sprint-driven development methodology and were the focus of a major empirical study in the socGSD project (see section 3.2 for more on the research context). As members of the socGSD project, our initial contact with the PyPy community happened in August 2006 when they organised a sprint at the university where we were active. It was organized in collaboration with another research group, but the circumstances were such that they could not themselves attend. Instead, the research group passed on to us the invited to join the event. At the time, PyPy was partly funded as an EU project and the community were very keen on having researchers study their work, since it was part of their EU mission to demonstrate the applicability of sprint-driven development in an open source context. This presented us with a very convenient opportunity for a case study of the practices of an open source community. The sprint-driven methodology made them an even more interesting case, as it was a novel approach to dealing with the complexities of distributed software development. Thus, the study was an opportunistic, exploratory case study with a focus on collaborative work practices in the PyPy open source community.

In this section, I shall present the PyPy case study and the analytical categories that emerged through it. The purpose is to provide an understanding of the details of the setting and to explain PyPy’s practices of sprint-driven development. Because it was an exploratory investigation, the description of this study will put more emphasis on how the research process than the preceding one. I will explain how our research questions evolved during the study, as this provides important insights into what characterised the

---

15 The study of PyPy took place within the socGSD group and in my presentation of the case I shall use the personal pronoun “we” in reporting on our joint research activities. However, it should be noted that I had the leading role in the study, and the specific analysis presented here is my own work. Having said that, I am in great debt to my friend and colleague Anne Sheehan from the socGSD project for her assistance with the content of this section, and indeed with all the material from the PyPy study presented throughout this thesis. We collaborated closely during the study and Anne shares an equal amount of credit for the results of it. Her creative and insightful thinking contributed significantly in the development of the analyses presented here, and the text itself has benefited greatly from all her helpful critique and assistance in writing it.

16 [http://codespeak.net/pypy](http://codespeak.net/pypy) (Accessed 2010-02-05)
PyPy community's work practices.

4.3.1 The PyPy open source project

4.3.1.1 An open source community developing a Python interpreter

The PyPy project revolves around developing an implementation of the open source programming language Python. Python is an interpreted programming language, meaning that code is executed by an interpreter. The de facto standard implementation of Python is CPython, which is an interpreter written in C. PyPy's implementation, however, is written in Python itself. Hence the name “PyPy”, which is an abbreviation for “Python in Python”. Another key part of the project is a translation framework which allows any program written in RPython – a subset of Python defined in the PyPy project – to be translated into lower-level languages, including C, Common Language Infrastructure (CLI), and Java. This means that PyPy's high-level implementation of Python – the interpreter – can be effectively customized and targeted for a number of different platforms. At the time of writing (January 2010), PyPy is at version 1.1 and can translate RPython programs (including the interpreter itself) to POSIX, .NET and JVM, corresponding to the three programming languages mentioned above. Thus, the two primary outcomes of the project is a Python Interpreter and a Translation Framework, which together produce effective virtual machines for Python on various target platforms. Additionally, the project has a broader goal to contribute to research on theoretical approaches and solutions for flexible and modular language platforms. The members of the community are actively publishing their work in relevant research forums.

PyPy is in many ways similar to the typical open source project. The community maintains a webpage with extensive project documentation, as well as mailing lists, chat channels, and an online code repository. Anyone is free to participate online and to download source code, either for studying it and developing new modules or functionality, or for compiling the product to use or distribute it. In principle, anyone is

17 This section provides a brief look at the PyPy community and project, but for more details see During (2006a, 2006b) and Sigfridsson, Avram, et al. (2007).
18 http://www.python.org/ (Accessed 2010-02-05)
19 Portable Operating System Interface (for Unix), Microsoft .NET Framework, and Java Virtual Machine.
also allowed to contribute code to the repository, with both a community review procedure and a version-control system (they use Subversion, SVN) ensuring that no one breaks the code. But in some respects PyPy stand out from the general open source crowd. Between January 2004 and May 2007, in order to accelerate the development of their product, they applied for and received funding from the European Union (EU). Following the efforts of a core group of community participants, the structured and rather rigid EU framework was adjusted to support the work of this *ad hoc*, flexible Open Source community. One of the key objectives of this setup was to support the use and demonstrate the applicability of a sprint-driven development methodology in a open source context.

### 4.3.1.2 Sprint-driven software development in PyPy

Building on principles and practices from agile development methodologies such as Scrum and eXtreme Programming (XP), a sprint in this context is an event where people otherwise working together from geographically distributed locations gather in one location for a short period of time and work in pairs or small groups on selected development issues. It is a time of rather intense collaborative coding and testing, but also for socializing with people whom one would otherwise only interact with online. This type of event has become popular within some open source communities – for example, the OpenBSD and Linux communities – and has many names, such as “hackathon”, “codefest”, “sprintathon”, to mention a few. The practice of using sprints for pivotal development was initiated by the Zope Corporation in the early days of the Zope 3 project.

One interesting aspect of the emergence of the PyPy community is that, in contrast with many open source projects, it didn't begin with an existing code base. The concepts and ideas that inspired the PyPy project evolved from mailing list discussions held in late 2002 among developers of the larger Python community. As During (2007) put it: “Inspired by but also frustrated with the results of two successful Python implementations, Psycho and Stackless, the ideas of a highly flexible and fast Python implementation purely written in Python was born”. The project itself then came into being as a result of architectural discussions held during a one-week kick-off sprint held in Hildesheim, Germany, in February 2003. As illustrated by the excerpt below, the
Some people are claiming that I caused this project to start off. Actually, it was Holger who started it, and this is something I appreciate very much. Without his sudden inspiration to just say "how about a sprint", I would never have had such an idea, and I'd never taken the possibility into account that such a project might become doable at all.22

This initial sprint enabled key design discussions to take place and set a direction for what would become the PyPy project. Sprints were thus an integral part of the PyPy project from the start and the idea of applying for EU funding was based in looking for ways to facilitate systematic sprinting. Indeed, once secured, the EU funding did allow PyPy to sprint more regularly than otherwise, with sprints every 6 weeks or so. This was a way to accelerate development as it provided regular intervals of intensified activity in the project, but importantly also to provide a forum where core members of the community could discuss intricate technical details and design decisions face-to-face. Sprints were organised in different locations to accommodate as wide a group of participants as possible, but also for the purpose of attracting newcomers to the project. Since most of the community was based there, sprints would for the most part be organised around northern Europe, but there were sprints on other continents as well, notably Asia and America. Part of the strategy was to also organize sprints in relation to larger Python-related events, because these were forums were many people with relevant competence would be present. For example, the Europython conference was – and is - a favoured opportunity for sprints.23

What we have begun to discern here, and which will become even more evident in the following, is how the regular collocated events were effectively interwoven with the dispersed collaborative activities of PyPy, as well as with all the individual work devoted to the project. This sprint-driven development process very much defines the PyPy project.

23 Please note that even though the description here is in past tense, the PyPy project is still ongoing at the time of writing (January 2010) and sprints are still a part of their activities. However, past tense is used because the period of interest was during the EU funding, which ended in April 2007. While the project is very much active and they do still sprint, it does not happen as regularly or in as organized a manner as during this period.
4.3.2 Research process – an exploratory case study

The PyPy case study went on for an extensive period of time, from August 2006 until September 2008. During that time we studied the PyPy community's activities both online and at collocated events, utilizing participant observations, in situ interviews, an ethnographic experience, qualitative analysis of online discussion forums, and open-ended email questionnaires. In retrospect, we can say that the research went through four major phases, which are summarized in Table 3 above. This sub-section will describe the details of this process.

4.3.2.1 Embarking on an exploration

The study began, then, as the seizing of an opportunity. PyPy organized a sprint at our university and we were invited to study them. The time given for preparation was a matter of weeks and since we had not studied open source projects previously, we began by surveying the current research landscape and reading up on the PyPy projects history.
One of the members of the research group had experience from studying open source development in previous work and she was able to direct our survey of what research had been done in the area, while Google searches led us to resources related to the PyPy project. Interestingly, among the most useful resources on the PyPy project were texts written by themselves and published in for us relevant academic conferences (e.g. During 2006a; During 2006b).

The approach was one of exploration and iterative refinement of research questions. Our actual empirical activities in this phase involved direct observation of the sprint (7 days long) and a survey of online forums, primarily the main mailing lists used by the community. At the sprint we were allowed to sit in to observe all the group meetings (which were also video recorded) and also to have in situ conversations with the participants at any time during their work (just to note, none of the researchers was involved as an active participant in this sprint). Based upon our interest, a PyPy project administrator agreed to organize a workshop on the PyPy development methodology, which included an extended discussion session with one of the founding members of the community. The study of online activities at this time served mainly as a way of gaining insights into the dynamics of the community and to build up a historical account of the projects development. We began following the main mailing lists a few weeks before the sprint and continued to do so after the event, up to the point were this phase of our study was concluded.

To kick off the analysis following the sprint, the whole socGSD research group spent
one day together reviewing the recorded material, comparing notes, and drawing up broad lines of interpretation. Subsequently, two of the researchers (myself included) took charge of continuing to develop the emerging interpretation. We drew on literature familiar to us within the project, attempting to explain what we had observed from a practice-based perspective on knowledge (Brown and Duguid 1991; Cook and Brown 1999), since this had been one of the main topics in our study from the outset. Consolidating our propositions with empirical evidence (particularly in terms of evidence in the online forums for practices similar to what we had observed at the Limerick sprint) and continuing to check our emerging interpretation through discussions in the socGSD research group, we began writing a conference paper on this topic. The first phase of our study culminated with that paper being submitted and accepted for the Open Source Systems conference in January 2007 (Sigfridsson, Avram et al. 2007). In this paper, we argued that sprints facilitated situated learning (Lave and Wenger 1991) and the attraction and enculturation of newcomers in the PyPy project.

4.3.2.2 Gaining an insiders view

When continuing on from the first phase of study, our perception was that the enculturation of new members was a key aspect of the PyPy community's sprint-driven development. Because of our orientation towards ethnographic principles, we considered getting an insiders perspective on what sprints meant to people joining the community as essential. A first-hand experience of the process, we posited, would help

---

*Figure 5: Images from PyPy sprint in Hildesheim, Germany, March 2007.*
After having consulted the core members of the community, I was allowed to join PyPy as a participant. Just as a regular newcomer, I began to study the technical documentation, familiarising myself with the source code, and participating in the community online. In March 2007, I attended a sprint in Hildesheim (Germany), taking the role of a regular newcomer and working on the project. The duration of this particular sprint was 3 full working days. The main method of record during this event was extensive note-taking – including both ongoing field notes and writing a fieldwork diary at the end of each day – and audio recording of all group meetings (mainly the daily scrum meetings). Apart from this, I had the opportunity for numerous casual conversations with the other participants about things experienced and was allowed to document much of the time spent there with photos. Added to the documented material was also the regular sprint report written by two of the core community members after the sprint. (A detailed account of this ethnographic experience is given in chapter 5.)

During this period we also assisted the PyPy community with some of their own efforts. They worked on texts about their own work practices and we were asked if we could help with this, because of our perceived expertise with academic writing. One of the researchers was asked to be a reviewer for one of the EU project deliverables. It was due on March 30th and documented the development process. This was important for our relationship with the PyPy community, as we were able to give something back to them, making it a relationship based on exchange. But it also helped further our understanding of the context of their work, as the production of EU documents was a vital part of the communities activities.

4.3.2.3 Looking for the constutively specific

The focus on newcomers had now become central to us. The participatory experience gave us very detailed accounts of the practices in PyPy related to newcomers. But we also realized that we needed to more systematically analyse the online forums. We were only seeing a subset of PyPy's activities at the sprints and also needed to counterbalance the ideographic nature of my participant observation. We began working on this track in April 2007, identifying newcomers and analysing the evolution of their interactions within the community. Because we were doing purely qualitative analysis
and the workload we were under at the time, we focused primarily on the development and sprint mailing lists (pypy-dev and pypy-sprint). PyPy's archives are organised to such an extent that it is easy to browse and to follow specific threads or persons. However, the material was still very vast and it was a cumbersome task to access it and keep track of progress in a sensible way. In the face of this challenge, we began employing grounded theory coding techniques, whereby we would read through the material and code – or mark-up – sections of interest with some concept or categorical label. Related sections of interest would thus be sorted into the same analytical categories. While still having to be performed manually, this helped us break down the material and identify what parts of it were interesting for our particular line of inquiry.

As our work progressed and we began discussing the analytical categories that emerged through the coding, we came to a realization. There was an obvious attempt at creating a continuity between online activities and sprints in the introduction of new members. People approaching the community online were directed to relevant documentation (both on system architecture and on work practice) and encouraged to discuss their intentions in the online forums. This served as an introduction to the technology and the people involved, which then in turn was an important basis for participating in a first sprint. The key of this realization was that the participants made no significant distinction between dispersed collaboration and collocated events, and not just in the introduction of newcomers – in practice, these two spheres were effectively interwoven, which is something that permeated the entire organisation and all activities of the project (more comments on this interweaving of online and collocated activities can be found throughout section 4.3.3 and in the conclusions in section 4.3.4). This roughly coincided with us presenting the first paper at the OSS conference in June 2007. We received quite constructive critique regarding our use of situated learning theory and in connection with also realizing that our interest was broader than just the enculturation of newcomers, we began to substantially re-think the purpose of our study of PyPy.

Focus therefore broadened to what role regularly occurring collocated events played in the overall collaborative effort. Simply put: why was it valuable to organize sprints in this otherwise well-functioning dispersed project? Our analysis was re-aimed towards identifying constitutively specific parts. In other words, identifying what could not be there if it was not for the existence of sprints. From having been introduced as a way of
dealing with the vastness of material, the grounded theory method of analysis now came to pervade our entire work on addressing these questions. That is, what became of importance for us was finding evidence for what was constitutively specific in the empirical material rather than according to some predefined theoretical perspective that was to be tested (e.g. situated learning). The coding process continued and our scope extended to encompass the time from the PyPy community's inception up until the point when we were doing the analysis. Our activities continued to focus on the mailing list archives, but also included going back to the material we had produced through observations, participation and interviews in our earlier work on the case.

Our coding procedures corresponded to the three phases that Strauss and Corbin (1990) termed open, axial, and selective. We had already initiated open coding – that is, breaking down the material by creating codes that conceptualise what had been observed - in our investigation of newcomers joining the community and could continue by simply extending our efforts outside the threads we had followed then and including our other empirical material in the process. As we progressed, we also began identifying connections between the various codes that emerged (axial coding) and a macro pattern with larger groups of related codes took shape. With selective coding, the researchers identify core categories around which the codes can be organized as support. In our case, the largest groups of related codes became significant analytical categories which we had a large amount of collected evidence in relation to; whereas codes that contained little material and did not belong to a large group could be discarded on account of our empirical material not being sufficient to support them. These analytical categories were given labels that described the common meaning across the codes they contained. It should be noted that even though these different phases were clearly distinguishable in our work procedure, it was not a matter of a distinct chronological sequence. This happened in parallel as the work progressed and we iterated between the different levels of analysis. Eventually, in late spring 2008, four significant analytical categories had emerged (these will be discussed in detail in section 4.3.3):

1. Synchronous work and project rhythm
2. Facilitating stronger social relationships
3. Attracting and enculturating new members (in correspondence with our previous
4. Deferring complex or controversial discussion

What we had now achieved was a proposition about the role of sprints in this particular dispersed FOSS community, grounded in empirical material.

4.3.2.4 Ascertaining the participants perceptions

However, while our analysis had included extracting participants own views on the meaning of sprints by identifying both discussions and singular comments where this was implicitly or explicitly expressed, we also realized there was a need to validate our ideas according to how they resonated with the participants own perceptions. This was somewhat of a common-sense realization, but it did echo the idea of interacting with participants for the generation of data which was central in our methodological background. Formal interviews would have been difficult to organize and would have risked disrupting the relaxed atmosphere that characterized our relationship with the community members thus far. Instead, we decided to use the medium which they felt most comfortable with – email – and design simple, informal questionnaires. It aimed at shedding further light on the analytical categories we had identified, without explicitly detailing them to the respondents, thus allowing for full freedom of response.

It was by now late summer 2008 and we got in touch with some of the core members of the community to discuss our intentions. The reception was enthusiastic and they helped us distribute the questionnaire to participants of a sprint that happened to be organized at the time. They also encouraged people to respond and in the end we got responses from both core members and newcomers. We included the responses in our coding and they helped confirm and, in some respects, refine our ideas. However, equally important to our research, was that the core members directed our attention to material on the role of sprints that they had produced. This included a video visualisation of contributions to the project source code repository over time, which showed how the sprints provided the fundamental rhythm of work in the project. It significantly contributed to our understanding of the role of sprints, as it made concrete some of the ideas we had been working on.
4.3.3 The role of sprints in PyPy

The research process outlined in section 4.3.2 led to four analytical categories regarding the role of sprints in the PyPy project. In this section, I will discuss each of them in detail.

4.3.3.1 Synchronous work and project rhythm

One of the key aspects of sprints in PyPy is that they provide regular intervals of intensified activity in the project. This does result in temporarily increased productivity, which is evident when studying how the number of lines of code increases significantly at each sprint.\(^{24}\) This was indeed one of the reasons why the core group considered sprint-driven development in the first place, as they believed that it would provide the momentum needed to achieve their ambitious goals in a comparably short time. It is also generally recognised within the community. For example, in a post on the PyPy blog that presents the latest visualisation of activities within the community, the general trend of sprint-related acceleration is highlighted:

> In the first part of the video, you can see clearly our sprint based approach: the video starts in February 2003, when the first PyPy sprint took place in Hildesheim: after a lot of initial activity, few commits happened in the next two months, until the second PyPy sprint, which took place in Gothenburg in late May 2003; around the minute 0:15, you can see the high commit rate due to the sprint.

> The next two years follow more or less the same pattern: very high activity during sprints, followed by long pauses between them; […]\(^{25}\)

The reason behind this rhythmic acceleration of development work is that there is a period of synchronous work at the sprints which intensifies the common effort. While the IRC chat channels do provide a forum for synchronous communication in the online setting, the sprints are the only regularly occurring periods where most of the core members are working together. The importance of this is evident in the emphasis that is put on actually getting as many full working days together as possible when the sprints are organized. Sprint planning is flexible to accommodate for unforeseen circumstances and ensure as much overlapping working time as possible. The conscious effort to maintain a continuity between online and collocated activities was also linked to the importance of synchronous work. The sprints are planned and prepared for in the


community's online forums, both practically and in terms of discussions that build up momentum on development issues that are likely to feature at sprints. This ensures that as little of the sprint-time as possible is spent on coordination, negotiation and setup, while as much time as possible is spent working together on solving actual tasks.

Furthermore, the agile techniques of daily scrum meetings and pair-programming contribute towards developers, not just being present for full working days at sprints, but to actually working together as much as possible during that shared time. This was also evident from our observations at sprints. At both the observed sprints, the importance of the participants being located in the same room and maintaining a certain awareness of what was happening in the other coding groups was noted. They made the decision to take a break – or continue an hour more than planned – by consulting each other. Peers were invited to have a look when unexpected errors occurred or a new solution was tried out. All of this was interpreted as efforts to maintain synchronization of effort within the group.

The increased productivity was thus a salient characteristic of the sprint-driven methodology in this particular context. Interestingly, looking at the matter in light of previous research findings, it seems like both the times of synchronous work themselves and the strong temporal rhythm into which these are organised contribute towards this. In a study of effectiveness in global virtual teams, Maznievski & Chudoba (2000) have found that “effective global virtual team outcomes are a function of appropriate interaction incidents and the structuring of those incidents into a temporal rhythm” (ibid. p. 489). They find that a strong, repeating temporal pattern of interaction incidents provide continuity and reduces ambiguity, and in doing so structures expectations and makes response times predictable (ibid. p. 487). This is in line with what we have seen in our empirical findings, were, for instance, the regular intervals of sprints allowed for a certain predictability. In PyPy, the sprints provided a very strong temporal pattern and the core community members are all adhering to this rhythm. The sprints themselves are bursts of activity, but the regularity of them also allows participants to work confidently and purposefully between them. This, in turn, means they can prepare – practically and mentally – and work towards the next sprint, so that they can get the most out of that activity.
4.3.3.2 Facilitating stronger social relationships

When interviewed about the successful evolution of the PyPy community at the 2006 Limerick sprint, one of the core developers responded by saying that “at the foundation is the relationship between people.” When the community was formed in 2003, the founding members had a shared goal and were driven by mutual interest and professional experience. But one of the most striking aspects when observing a sprint is the close relationship between participants and the sense of fun and enjoyment they derive from these opportunities to meet. As participants arrive, even as they set up their laptops, there is a constant flow of work related chatter, interspersed with laughter, and movement to-and-fro as they share screens and update each other on technical progress. As the project administrator stated during an informal interview (Limerick 2006), the core members of the project are all very comfortable with each other, like “soulmates” (her word).

A close personal connection has evolved in the group. Whilst the community initially intended to use the sprints as a way to accelerate development, it rapidly became apparent that the community members also wanted to take time to socialize together informally at the sprints. It has since become an accepted practice that a break day will be included as part of the sprint organisation when the participants can relax and take some time-out together.

Furthermore, the social aspects of the sprints also play an important role for the inclusion of newcomers into the community. Getting to meet ones fellow developers face-to-face was in fact one of the most cited reasons by newcomers who had joined the community. For instance, these quotes are excerpts from the questionnaires that we distributed to the community:

1) I chose to go to the Duesseldorf sprint in May-June 2006 because I was invited by the other developers; I wanted to do more work for my thesis and also to meet in the real life all the other people I was used to talk with by IRC.

2) I wanted to meet all those people that I had only interacted online with to that point. Also, I wanted to pair-program with some people, which is rather hard to do via the web (but possible).

So it is recognized that this focus on building and maintaining relationships is of benefit to the community. Our conception is that the sprints provided an important opportunity
to further strengthen both the professional and personal bonds that emerge from the online collaboration in PyPy, which was a vital contribution to the coherence of the community.

### 4.3.3.3 Attracting and enculturating new members

The great emphasis on sprints as a way of introducing newcomers to the project and helping them to get started is clearly visible when studying the PyPy community. As has been mentioned previously, sprints are organized in different locations to accommodate as many participants as possible and one of the purposes of this is to attract newcomers as well. When asked for the reason why they chose to go to their first sprint in our questionnaire, many emphasize this as a major reason:

1. The sprint was organized in ---- where I at that moment just started my PhD. The project idea sounded cool and I wanted to learn about the Squeak VM implementation as well as PyPy.

2. It was in my home city, so it was quite easy to attend. I didn't want to pass up this great opportunity to take part in this interesting project.

Part of this strategy is to also organise sprints in relation to larger Python-related events, because this is a forum were many people with relevant competence will be present. Consequently, there are many examples in the mailing list of newcomers who are attending conferences where PyPy sprints are being organized and want to participate in the PyPy sprints as well:

> I will be attending the post-EP [the Europython conference] sprint and wanted to quickly introduce myself. I’m a student at the University of ----, Switzerland, and was accepted in Google's Summer of Code (SoC) program [...] 

Another important part of this strategy is how the sprints were advertised. Significant efforts were aimed at reaching out to a wider group of potential participants by attempting to make sprint notices visible through many channels. For example, PyPy sprints are advertised in relevant forums of the larger Python community for this reason.

However, since the projects inception, there has also been a growing recognition that distribution of the announcement wasn't the only requirement for inspiring new participants. The technical complexity of the PyPy project presented a major threshold. To alleviate this difficulty, the PyPy community has built up extensive online
documentation - covering both technological details and development methodology - and this plays an important role in the reception of newcomers to the community. There are many cases of people approaching the community and asking for advice on how to get started. One of the first recommendations they will get is to have a look at the documentation.

Another important aspect of the reception of newcomers is how they are also immediately invited to ask questions and to begin discussing their intentions at the available online forums: the mailing lists and the IRC chat channel. So reading the technical and methodological documentation, familiarizing oneself with the source code, and beginning to discuss ideas and intentions with the current members was considered essential introductory steps for people approaching the community online. Furthermore, this was not separate from the sprints as a way of introducing new members to the project and community. The interweaving of the collocated events and the online activities was apparent, because these initial activities were in many ways seen as preparation for the sprints.

This was obvious in our analysis of newcomers interactions with the community in the online forums. If a person who began to participate online showed good skill and a willingness to contribute to and invest substantially in the project, the question that would inevitably be asked was: “Will you be able to make it to a sprint?” Well at the sprints, the newcomers were effectively taken by the hand and there was a big effort to provide support for them. Tutorials were organized and newcomers were allowed to team up with experienced developers for pair-programming. There was also an emphasis on letting newcomers work with as many regular participants as possible, and the pairs would regularly change for this purpose.

In relation to this, one key aspect that was observed during the participant observation of the Hildesheim sprint was the informal mentoring that goes on at sprints. This arises from the fact that most of the core developers are likely to be present at the event to offer support and advice. One of the practices that was observed was core members “wandering” - they would simply spend a lot of time walking around the room as people were working. Sometimes they would stop and ask someone about the work or to offer some advice, but a lot of the time they seemed to do this as a way of making themselves available. Through this behaviour they would remind people of their presence and show
The importance of quickly marshalling and continuously sustaining the community behind an open source project has been emphasised by researchers (e.g. Krogh, Spaeth et al. 2003; Ducheneaut 2005; Ye, Nakakoji et al. 2005) and I also highlighted it as one of the key topics for exploring change in distributed software development projects (section 2.5.2.2). As we have seen in this section, the sprints played a significant role in creating and sustaining the PyPy community. The key point is how the PyPy community's practices for receiving newcomers focused on enabling them to achieve both technical competency and community membership. It is also important to note how the online activities and the collocated events were interwoven in this process.

As part of addressing the issue of how software developers in distributed projects develop their practical knowledge as they deal with changing circumstances, I will explore PyPy's practices related to the introduction and enculturation of newcomers further in one of the vignettes of chapter 5.

4.3.3.4 Deferring complex or controversial discussions

As we saw in section 4.3.3.1 above, sprints provided focused and intensified periods of activity in the PyPy project. This was one of the key motivations for the adoption of a sprint-driven methodology. Whilst, in common with other open source communities, various computer mediated communication channels are utilized by the PyPy community, the sprints also provided regular opportunities for face to face discussion time for the core members of the community. The community instigators came to realize the value of co-located meetings, particularly for in-depth discussions about intricate technical details and design decisions:

*I think the sprint was very productive. There were quite a lot of design questions that have been sorted out by the whole group discussing together. Doing the same on mailing lists would have taken much more time and misunderstanding (as e.g. might be the case now on the stdobjspace thread).*26

As evidenced by the response to questionnaires, the opportunity to meet face-to-face fostered creativity and innovation beyond just the immediate work task in a way that was difficult to nurture in online discussion forums, where the conversations by nature

---

of the medium require more structure and clarity:

The sprints are usually periods of extremely productive programming and probably even more importantly, discussions. The sort of imaginative brainstorming you can have at sprints (particularly in the evenings over dinner) is very hard to get online. Therefore going to sprints usually brings me forward in many ways, usually getting more clarity and new ideas for my CS research and my PyPy project work.

The value that PyPy found in this can also be seen in how the sprints were organized in a manner that encouraged participation in group discussions and information sharing sessions. There were, for example, daily scrum meetings, but the community also refined their rules of conduct for such purposes, as we can see in this post-sprint reflection on their work practices:

Also Armin and me discussed on IRC that it would be nice [...] b) to set up some "organization" rules like e.g. when we gather to tell about achievements-of-the-day that everyone actually listens and stops hacking :-)  

However, again it must be pointed out that this was not distinct from the communities online activities. Rather, the point is that these collocated discussions provided an additional dimension to ongoing discussions within the community. The topics of online discussions online would become the focus of sprint work. And while the sprints provided an opportunity for intensified discussion and clarification of design issues, these discussions would in turn flow onto the online forum spaces when the participants dispersed. Furthermore, the opportunity to meet in person actually enhanced subsequent ability to communicate online, as was pointed out in another questionnaire response:

Online interaction becomes a lot easier when you have actually met several of the involved people and know them well. IRC is a medium that is prone to misunderstandings and in my opinion knowing the involved people lessens that risk.

Our conclusion is that having interceding online and collocated discussions provided the PyPy community with the ability to defer complex or controversial discussions from online forums to face-to-face meetings. The resulting interweaving of discussions in various spaces strengthened the participants common ground. I will return to this particular aspect of sprints in PyPy in a vignette in chapter 5 that explores an episode where the project goals were being negotiated.

4.3.4 Summary and discussion

This section has presented an exploratory case study of an open source community called PyPy. It was a study within the socGSD project that aimed to examine how a sprint-driven development methodology was employed by the community as a way of dealing with the complexities of distributed software development. Methods included participant observation, in situ interviews, qualitative analysis of online material, open-ended email questionnaires, and an ethnographic experience of actually joining the community and attending a sprint. I summarise the findings in text below, but see also Table 4 on page 88 for a condensed summary.

Through the exploratory investigation – which ranged over two years – four significant analytical categories emerged that explained the role of sprints in PyPy. Firstly, they created a fundamental rhythm for the project, which, in combination with the intensive burst of synchronous activities that sprints entailed, contributed to increased and sustained productivity. Secondly, regular collocated meetings facilitated and strengthened social bonds among participants. Thirdly, sprinting served as an essential process for sustaining the community, whereby newcomers were attracted and enculturation facilitated. And fourthly, sprints served as an important forum for intricate discussions, and the presence of sprints therefore presented the community with the ability to interweave discussions in different spaces, something that was important for maintaining coherency among community participants.

There are two key conclusions to be made from the PyPy study within the context of this thesis. First, that their practice of sprint-driven development – of interweaving online activities and collocated events – was their way of achieving the complex task of distributed software development, but not because it was a specific way of dealing with the issue of “distribution” or “dispersion”. Rather, it was so because it fulfilled more specific purposes within the community, which were framed in our analytical categories. Secondly, that many elements of PyPy's practices had evolved because the community expected some things to keep changing, knew that some conditions would not be stable. Particularly, that there would always be people joining and drifting away, and that the vision of the project was under constant negotiation. It is important to point out, also, how it was shown that the PyPy community learned over time the value of sprints in relation to these issues. There is important evidence for my argumentation.
about the purposeful adaptation of practice here (see chapter 7) and I will also be exploring these aspects of PyPy's work further in the following chapters (5 and 6).

<table>
<thead>
<tr>
<th>Analytical category</th>
<th>Findings</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous work and project rhythm</td>
<td>· Sprints provide periods of intensified activity within the community</td>
<td>The sprints themselves are bursts of activity, but the regularity of them also allows participants to work confidently and purposefully between them, hence sustaining a high level of productivity.</td>
</tr>
<tr>
<td></td>
<td>· Regular sprints provide a work rhythm within the project</td>
<td></td>
</tr>
<tr>
<td>Facilitating stronger social relations</td>
<td>· To meet face-to-face at sprints is a driving factor in the community</td>
<td>Sprints provide important opportunities to strengthen professional and personal bonds.</td>
</tr>
<tr>
<td></td>
<td>· Much effort goes into socialising with each others at sprints</td>
<td></td>
</tr>
<tr>
<td>Attracting and enculturating new members</td>
<td>· Regular sprints at various locations and venues are a strategy for attracting newcomers</td>
<td>Sprints are important for sustaining the community.</td>
</tr>
<tr>
<td></td>
<td>· Newcomers are taken by the hand at sprints</td>
<td></td>
</tr>
<tr>
<td></td>
<td>· The sprints and the online activities complement each other</td>
<td></td>
</tr>
<tr>
<td>Deferring complex or controversial discussions</td>
<td>· The opportunity for collocated discussions among core members are highly valued</td>
<td>Sprints are a way to work out uncertainties and strengthening the common ground of the core group.</td>
</tr>
<tr>
<td></td>
<td>· Online discussions and collocated discussions intercede each other and are not distinct</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Summary of findings from PyPy case study.
4.4 A comment on the comparison of organisational and open source software development activities

We have now seen the results from two different field studies of distributed software development. It is obvious that the circumstances under which people worked in the two different projects described were vastly different. The Lambda project was organised top-down, with given objectives and resources, whereas PyPy was voluntary and free-flowing. However, when we look at the details of their practices, we can see that they were engaging in the same substantial activity - distributed software development.

A software product had to be engineered, which in both cases involved engaging in related practices for collaborative software design, coding, and testing. This effort had to be performed by collaboration between geographically separated partners, making communication and coordination across distance vital. Albeit different strategies were applied to cope with this in both cases, we also saw that this dimension was not experienced as a substantive problem in either case. It was simply something to contend with as part of the daily work activities.

Importantly, we also see how the work flow in both cases was defined by the necessity of dealing with evolving purposes and goals. In PyPy, the sprint-driven methodology created a basic rhythm and enabled them to negotiate their common enterprise and sustain the community. For the Lambda team, applying the agile method Scrum enabled them to develop the product despite the fact that the long-term project plan was not specified and that requirements kept changing due to organisational decisions beyond their control.

As has been mentioned previously (see section 1.2 and 3.4.4 in particular), the purpose of this thesis is not to evaluate the differences between distributed software development in organisational settings with that in open source settings. The point I want to show with this section, is rather that there were fundamental similarities in the challenges facing participants of the two projects. This is the reason why I treat the Lambda and the PyPy cases as comparable. Studying practice in these two different settings allows me to show the development of practical knowledge through dealing with changing circumstances as a significant problem to distributed software development as an activity across different contexts.
4.5 Summary of chapter findings

This chapter has detailed the two field studies which constitute empirical foundation of this thesis: an observational field study of a software development team in a multinational organisation, and an exploratory case study of an open source community called PyPy. The first shed light on the work practices of a team working in a distributed project. The second on the practices of sprint-driven development in the context of a dispersed community. (Sections 4.2.4 on page 67 and 4.3.4 on page 87 summarised the findings of each study respectively.)

First of all, the combined image of distributed software development practice that has emerged shows the distributed dimension as a mundane issue, or how the complexities of distributed software development were dealt with as a matter of specific workaday problems. In the Lambda case, it appeared alongside many other equally significant issues. Being part of a global project – and indeed a global organisation - was not considered as a distinct issue. In PyPy, the interweaving of online activities and collocated events served more specific purposes within the community than simply alleviating distance.

Secondly, in correspondence with the perspective on practice assumed in this thesis (discussed in section 2.5.1), issues related to emergence and change were found prominent in both cases. The Lambda study provided evidence on the prominence of continuously changing circumstances and gradual adjustment of practice. Likewise in PyPy, which provided additional insights about practices intended specifically for dealing with emergence. These particular findings are important for the purpose of my thesis, which is to investigate how practical knowledge is developed in dealing with changing circumstances. Having provided an empirical context and this initial evidence, the next chapter will explore these specific aspects of the work in more detail through the topics of evolving technology, enculturation and socialisation, and emerging project purposes and goals (suggested as relevant in chapter 2).
5 Exploring issues of change in technology, participants, and goals

5.1 Introduction

This chapter explores in detail issues of evolving technology, enculturation and socialisation, and emerging project purposes and goals within the two settings under study in this thesis, the Lambda team and the PyPy community. With this thesis, I am addressing the issue of how practical knowledge is developed in distributed software development practice by looking at how software developer groups deal with changing circumstances. The three topics explored here are relevant venues for investigating changing circumstances in distributed software development practice that I suggested in chapter 2 (see particularly section 2.5).

I elucidated practice in the Lambda team and the PyPy community with analytical categories in the previous chapter (number 4). In this chapter, I build on that material to provide more in-depth analysis on how the three topics related to change manifests in those particular settings. I do so by rendering four vignettes – two from each setting - that illustrate episodes where there was change in technology, participants, and goals in these projects. The vignettes are rendered from the significant lines of inquiry during my field studies (section 5.2 below elaborates on how they were selected and discusses their representativeness).

The episodes are presented in narrative form, emphasising the sequence of events and activities in each episode. My intention with this is to provide an explanation building analysis (Yin 2003) of the individual episodes, tracing out the chronology of what happened and stipulating a set of presumed casual links. This allows me to analyse what
constituted the practical problems in each episode, refining our understanding of them in the context of distributed software development practice. The outcome is additional empirical evidence in relation to the main objective of my thesis. In the following chapter (number 6), I will discuss what the common themes are between the different episodes and propose a characterisation of how changing circumstances unfold in distributed software development practice.

5.2 Selection and representativeness of the vignettes

The episodes described by the vignettes in this chapter emerged as significant during my field studies and as central in my analysis (details on methods in each study in sections 4.2.2 and 4.3.2). This was due to me adhering to grounded theory principles and techniques, as well as the ethnographic commitment to attempt to see the world from the perspective of those studied (see chapter 3 for more on my methodology).

First, when immersing myself with the software developers under study and otherwise generating empirical material on the cases, certain events and topics were raised as important by the participants, either in conversations or as observable in what they were devoting their time to (in terms of actions and discussions). Episodes relating to these are significant because the software developers themselves ascribed them a high value and they played a large role in their working experience at the time. Furthermore, I had to choose to follow certain lines of inquiry during my field studies, simply because an observers capacity is limited. Episodes related to the events and topics that were raised as important became the main stories that I focused generating empirical material on.

Second, in my analysis of empirical material using grounded theory coding techniques, the four particular episodes in this chapter were well represented across the significant analytical categories that emerged (presented in sections 4.2.3 and 4.3.3). This means that I have a lot of empirical evidence on them, while they are also central in my overall analysis of the cases. It is on this ground that they were chosen over other potential episodes, for which I would only have partial empirical evidence and that would only represent certain elements of my analysis.

So the episodes presented here were among the main stories (events and topics playing a large role in the developers everyday work during the time of study) that I followed with my empirical efforts. They were selected because of their prominence during field
studies and that they were well-represented across the analytical categories that emerged through grounded theory coding. I thus had sufficient evidence to elaborate in detail on the chain of events and make analytical statements about them. This makes them representative from the point of view of my empirical findings. But their bottom-up emergence was also consolidated with relevant literature. In parallel with their emergence in the field and during analysis, I tuned my reading towards related issues. The four episodes presented in this chapter correspond to the topics introduced in chapter 2. This confirms that the episodes represent important issues, not just in my particular cases, but also within the domain of distributed software development. The four vignettes of this chapter are thus both empirically grounded and situated within the boundaries of the study area.

5.3 Vignette 1: Lambda development team dealing with technological contingencies

This first vignette illustrates an episode where the Lambda team had to adapt the product under development following an update of the software framework it built on. The purpose is to show how the topic of rapidly evolving technology as a source of change manifested in the particular context under study (a topic introduced in section 2.5.2.1). See also Figure 6 below for an overview of the sequence of events depicted in this vignette. It is also recommended that the reader consults 4.2.1 and Figure 2 on page 56 (which shows the sites and relationships of the Lambda project) to get a sense of the context in which these events took place.

![Figure 6: Sequence of events in vignette 1.](image-url)
5.3.1 Lambda development team deciding whether to update or not

First, let's begin with a look at the context of these events. The Lambda team worked on a product that was already released and in use by customers. Their work was mainly aimed at developing the next major version of this product (I will call this “Product 2.0”, as compared to “Product 1.0”, which was the version on the market). However, they were also dedicating some time to incrementally updating the released version (such updates will be referred to as “1.x patches”). So while the main effort was aimed at developing Product 2.0, some periods were dominated by finishing 1.x patches so they could be distributed to customers. Another vital piece in this story is that the product being developed constituted a set of portlets for an enterprise portal software framework (this will be called “Framework”). That is to say, the Lambda development team developed software pieces that worked within this larger software system. The software framework, in turn, was autonomously developed and maintained elsewhere in the organisation.

In early November 2007, some of the Lambda developers were working on completing a new 1.x patch. They were aware that a new version of the software framework was then currently in beta and due for release soon (I’ll refer to the new version as “Framework 1.1”, an update from “Framework 1.0”). But since they had tried Framework 1.1 briefly, they knew of a few issues with this early version and had decided not to support it in the 1.x patch. During a work session where Framework 1.1 was tested, it was argued that, since the 1.x patch had to be completed only the following week and there were these known issues, they would be “better of going back to [Framework 1.0], cause we need this up and running” (Field notes 2007-11-07). An early build of the 1.x patch was created on that same evening, without support for Framework 1.1. It's noteworthy that an announcement about this build was also distributed on the project mailing list, because the project release manager, who worked in a US office, replied to the notification mail. He asked a number of questions regarding the build, among them whether the new framework version was supported. A developer replied, explaining that they had run into problems with the framework beta version and had decided not to support it as the final release of Framework 1.1. was not due until the following week. They continued to work on the 1.x patch without support for Framework 1.1.
On Thursday the following week, the Lambda team had a team meeting where the approaching 1.x patch release was discussed. This was only two days before the 1.x patch was expected and the final version of Framework 1.1 had been released the day before. Regarding the newly released framework version, it was pointed out that it was not used by any customers at the time and would also likely make necessary significant changes in the 1.x patch. Therefore, the local program manager corroborated the decision not to support Framework 1.1 at the time, deferring its implement to the next development phase. The matter seemed settled.

However, on the afternoon of the same day, the projects release manager – located in the US site – reversed the decision not to support the new framework version. Suddenly, the Lambda team members who were responsible for the 1.x patch build had no choice but to address the challenge of implementing support for Framework 1.1. The deadline for delivery was not changed, so time was short.

5.3.2 Lambda development team dealing with the framework change

When trying Framework 1.1 with the 1.x patch previously, Lambda team members had seen that there were conflicts, but had not investigated in any detail the exact cause or effect. However, since then they knew at least that the issues related to the installation process, and so it fell to the developer responsible for this part of the product to investigate exactly what the issue was. He downloaded and installed the new Framework 1.1 and began to test running their 1.x patch on it, while also letting others know what was going on in an email: “I downloaded [Framework 1.1]. it's here if anyone wants to take it for a spin: [intranet address]” (Mail archive 2007-11-15). As it turned out, there was quite a serious issue that caused the installation process of their product to fail. The developer called on one of the technical leaders for assistance in trying to find the exact cause. They went about this by reading in detail the release notes – which was produced by the Framework development team (working in another of the organisations site) and documented changes in the new version – and by tracing the error in the code. When I spoke to the developer after the events, he described the issue as a “quirk” in the new framework version where vital parts for their installation process were missing.

Having traced the cause of the problem, the Lambda team members were able to work
out a temporary fix in the installation procedure which required some extra steps during installation on part of the user. Apart from the hours spent finding the problem and coming up with this fix, the team also needed to run all the build tests again. Not until late night on Friday was there a final build of the 1.x patch. Its announcement on the mailing list was accompanied by notes on how to use the fix.

Interestingly, in the aftermath of this event, it turns out that awareness of the change outside the group directly involved was limited. Even though instructions on how to use the new installation scripts had been distributed to the Lambda project email lists, some developers still expressed surprise at the following weeks team meeting when realizing their original decision about Framework 1.1 had been reversed. At a core team meeting (where leaders from each site met using a telephone conference system to discuss project status) that same week, a remote test team leader raised a problem they had found, only to be informed that this problem was only due to them testing the product using the old framework version and that it was no longer an issue since development of 1.x patches had shifted to support the newer version.

### 5.3.3 Lambda team members reflecting on the incident

There were also some ripple effects of these events during the following months. Particularly, I observed a number of discourses related to the framework update dealt with here, either in topic or by reference. In the team meeting a week after the 1.x patch, the Lambda team members explicitly discussed what needed to be done since they had begun supporting Framework 1.1. It was, at that time, not decided when they were going to implement support for this new version in Product 2.0, but they knew for certain that they must work with it for 1.x patches from there on. They discussed, for instance, how there might have to be new test environments set up in the project to allow for parallel testing with different Framework versions.

At a technical lead meeting in January 2008 the following year (two months after the main events of this story), a major discussion regarded general strategies in relation to updates of framework software. The events we have seen here were not explicitly mentioned, but the problem, as it was described, related to how they could “avoid having to download and support new drivers every week” (Transcribed notes 2008-01-23). They discussed in particular what version of Framework was currently the best. The
main architect pointed out that an explicit strategy with regards to such issues would require someone to monitor updates and make judgements about when to move to a new version, but that this was time and expertise they could not spare.

Around the same time – January 2008 – the Lambda team also had a lessons learned meeting (where the group could reflect on the work performed and how it could be improved) in connection with a scheduled development sprint wrap-up. At this meeting, the developers expressed concern that changes in product structure and framework software was not communicated properly to them. The particular incident they discussed was similar to the Framework 1.1 events we have seen here and referred to how another piece of software used in the product was updated without them being informed. The developers suggested that when design was changed, they needed to sit down as a group to discuss why and what it meant for the continued work. Interestingly, two days after these meetings, one of the developers sent a mail to the project mailing list with information about the latest updates of Framework.

5.3.4 Summary and discussion

What this vignette has illustrated is an episode where the Lambda team had to adapt the product under development to the – from a local perspective - autonomously developed software framework it built on. (See Figure 6 on page 93 for an overview of the sequence of events depicted.) This has exemplified the topic of evolving technology (introduced in section 2.5.2.1) within the particular context of the Lambda project.

What has been shown is how the episode unfolded through a complex interplay between actors involved, as predicted by literature (see 2.5.2.1 ). Since the Lambda project was distributed, this included both interplay within the Lambda development team and between various stakeholders in the project. The source of the problem was the technological contingencies of the product under development, as it was an intricate part of a wider software architecture and therefore had to be adapted to changes in this environment. The challenge of investigating the impact of a framework update on their product and working out any necessary adjustments fell on the Lambda team developers. However, a key turning point in the episode was when a manager in the US made a decision that imposed an immediate need to work out a solution. Furthermore, in the aftermath of the incident, awareness of the decision to support the new framework
version in the patch took time to spread, both within the Lambda team and among project stakeholders, even though the solution was communicated via the project mailing list. The technological and collaborative relationships thus apparent in this episode play an important role in my discussion of common themes between the different episodes in chapter 6.

Additionally, the events also influenced the subsequent work within the Lambda team, as they sparked reflection among the participants about issues of updates in technology they were dependent on. They had an impact on the team leaders thinking about strategies regarding external software updates, and also made an impression on the developers perception on how such issues should be communicated within the team. This is an important insight in relation to my argument about the purposeful adaptation of practice in response to changing circumstances (see chapter 7).

5.4 Vignette 2: The introduction of a novel collaboration platform in the Lambda project

We stay with the Lambda team for this second vignette. What is illustrated here is an episode where a new collaboration platform was introduced in the project and how the Lambda team in particular utilized it. It is a second example of the topic of evolving technology (introduced in section 2.5.2.1), particularly in terms of the intricate ties between technological evolution and changes in organisation and practice. Figure 7 on page 99 contains an overview of the sequence of events depicted in this vignette. It is also recommended that the reader consults section 4.2.1 and Figure 2 on page 56 (which shows the sites and relationships of the Lambda project) to get a sense of the context in which these events took place.

5.4.1 Introducing Accord to the Lambda project

For simplicity, but while also retaining anonymity for the study participants, I will give the system that was introduced to the Lambda project the pseudonym “Accord”. The Accord platform integrated support for many different aspects of collaborative software development, including source code management, task tracking, and project planning. When I began my observations with the Lambda team in late August 2007, there were already discussions about introducing Accord as the main collaboration platform in the project. The introduction was led by the Lambda development team - spearheaded the
installation and also maintained the system – although major decisions regarding it had to be taken in consultation with test team leaders and the US managers, since it was the infrastructure for collaboration within the whole project.

At a Scrum development sprint planning meeting in early September 2007, the Lambda team leader group were discussing general development strategies and plans regarding system architecture and work organisation. This was a planning meeting where the broad strokes of the development work for the projects second phase were re-drawn in light of recent organisational changes (see section 4.2.1 for a brief history). One topic on the agenda was the Accord system. Moving to this new collaboration platform was planned as an essential part of the projects future. It was the product architect who was the initiator of this ambition and he was championing the systems introduction. Another of the meeting participants raised the question of whether they were going to use Accord for source code management as well. The answer was that this was the plan, but though the issue was addressed, there was no committing response.

At a team meeting one week later, the developers were told that a server running Accord would be set up the following day. The circumstances related to its introduction was also explained. They were told that Accord was still under development itself and that

Figure 7: Sequence of events in vignette 2.
the Lambda project was one of the very first to begin using it in live production. Lambda also had contacts in the Accord development team and they had relayed an interest in getting comments and feedback from Lambda's experiences. Following this meeting, the Accord server was indeed installed and, as work on the next sprint began, the team began using Accord's task tracking and project planning services. However, the question of whether to also move the Lambda source code over to Accord remained an open one.

5.4.2 Contingencies prompting caution among Lambda team leaders

Almost a month later, in early October 2007, the Lambda team technical leaders devoted another meeting to discussing the situation with Accord. The main topic was summarized in the meeting minutes as: “We need to confirm our path forward.” (Collected artefact, 2007-10-10). I recount the discussions from this meeting in some detail, because they highlight the primary concerns that the team leaders had about moving the Lambda source code to Accord.

The problems mainly stemmed in the fact that Accord itself was still under development. Above all, the team leaders were afraid that major revisions and updates made in development, or crashes due to unresolved bugs and undiscovered errors, could make their source code unavailable. At the very least, they would have to come up with a reliable backup procedure, something external from Accord so that their data could be safe regardless of the platforms development trajectory.

One argument made in favour of Accord, was that the Accord development team themselves were using the system in their work (eating their own dog food, as it were). Presumably, this showed that the system was mature enough for live development. However, one of the meeting participants countered that there was a crucial difference. The Accord team had a completely different level of expertise and insight compared to the Lambda team. He meant that if the Accord team encountered a problem, they could simply go right into the code and fix it, while the Lambda team had neither the expertise nor the access to do so.

Additionally, there would be the technicalities of actually moving the data to the new platform. They would have to investigate how they might have to restructure the artefacts in such a process and what effect that would have on their automated work
processes. Also, they would have to assign people in the team to actually execute the move and learn how to maintain and operate the new source code management system. They had already assigned one developer to administer the Accord servers and he was deemed the most suitable candidate for such a task. However, the problem was whether they could spare even more of his time. It was pointed out that he already spent half his days managing the new servers, while the team at large was pressed for resources.

With these points having been made, the meeting ended. No decision was taken, the meeting having served more as a forum where the team leaders could air their concerns about Accord. What we have seen in this, though, is how introducing this new platform was an issue of bringing in additional contingencies to evolving technology. These contingencies continued to be a source of uncertainty in parallel with the activities that are described below. For example, at a team leaders meeting in late November 2007, one participant strongly argued that there was too much uncertainty – shifting focus from the old system to Accord, back to the old systems, and even rumours of another system, etcetera – and that the developers ("The guys actually living with it", as he put it) were feeling frustration about it.

5.4.3 Lambda team evolving shared practices for the new platform

To recap, the Lambda developers began using Accord for task tracking in their daily work during the later half of September 2007. There were no noticeable hiccups in terms of work flow, as everyone seemed able to pick up and work with the system from the start. Most had experience using similar systems in the past and even the ones who didn't said, when I brought the question up, that they had no problem grasping the basic concepts and using the interface. But from the point of view of the teams coordinated activities, some issues began accumulating as time went by. One week after the above mentioned technical leaders meeting – in mid-October 2007 – the project manager spent her day consulting each individual developer about how they were entering their tasks into Accord. I asked her about this and she told me that they were trying to encourage people to “clean up” what they entered into the system, so as to make it easier to maintain a project status overview.

However, at a technical leaders meeting two weeks later, the issue of people having incorrect or outdated task information in Accord was raised as something that had to be
dealt with. The problem had slowly escalated as time went by. The project manager relayed some of her observations about how people were using the system and the problem was portrayed as partly being one of habit. The developers were simply not keeping their tasks in Accord up to date. But it was also a matter of what information people are actually entering into the system, as they were categorising and labelling things differently. The participants at this meeting began discussing “best practices” (their words) and it was decided that the project manager was going to draw up guidelines. It was also decided that they would talk about the issue at the upcoming team meeting, which was scheduled for the following day.

At this meeting, the project manager devoted a session on the agenda to suggesting “best practices” for Accord. She explained the problem of people not maintaining task information properly and began with suggesting that people try to incorporate using Accord into their daily routines more, at least checking it first thing in the morning and last thing in the evening. She also mentioned the importance of categorizing their tasks correctly, as well as entering “time estimations” and “severity and priority”, even if they were difficult to estimate. There was a general discussion about how these “best practices” could best be “disseminated”. It was suggested that some fields be made obligatory – that is, that they must be filled in, ensuring that people enter all the important information – and that the other information should be put on a wiki page.

One particular detail that was interesting at this meeting was how one developer reacted with surprise when the project manager stressed the need for people to update task ownership properly. He asked if there wasn’t an automatic mail notification when someone traded tasks, something a few other participants agreed was common. But the developer who was responsible for administering the Accord server said that, at least there wasn’t any such function activated on their servers. After the meeting, I interviewed the developer who had reacted with surprise and he told me that he had simply assumed there was such a function, because there had been in a similar system that he had worked with previously.

During the two weeks following this meeting – late October to early November 2007 – there was a heightened level of reflection about Accord usage. For example, I observed many instances where developers asked the project manager for advice on how to label specific tasks. There were also instances where developers reported – in an informal
manner – to her about mistakes they had made. For example, one developer mistakenly opened a task and was thus assigned it in the system, but since he hadn't meant to, he had to close it again. Instead of letting the matter be, he called across the cubicle wall to let the project manager know about the mistake. I also observed an increased tendency to discuss Accord and how to use it during regular work sessions. Interestingly, discussions about what you could do in Accord often took the form of “[…]well, in Bugzilla you could […]” (Field notes 2007-11-21), i.e. referring to similar systems they had previously worked with.

In terms of team activities, the “best practices” for using Accord remained a topic at team meetings. Three weeks after the team meeting when “Accord best practices” were first brought up, the project manager said that she perceived some “improvement in how you're using [Accord]” (Transcribed notes 2007-11-21), but it was added that it was still “somewhat chaotic, with people having tons of open tasks, the same task being opened by several people, people working on stuff not in [Accord] at all, etcetera.” (Transcribed notes, 2007-11-21). That the problems remained was also confirmed another two weeks later, in early December 2007, when another long discussion about Accord dominated a technical leaders meeting. Leading on from this, and also confirming the continued issues with how people were using Accord, it was announced that an updated and extended version of the “best practices” wiki page was under construction at a team meeting two weeks later (mid-December 2007).

What this aspect of the story illustrates is that learning to use Accord was not simply a cognitive issue, but a collaborative problem. The main problem was conflicts due to differing practices and the team had to negotiate a common road for moving forward.

5.4.4 Co-adapting Lambda team and Accord system roles

In the meantime, another set of issues had begun to surface in parallel following the introduction of Accord. As was mentioned above, at a team meeting in late November 2007, there was a mention of how developers in the Lambda team were working on tasks that were not entered in the task tracking system. When this came up, some developers raised concerns about how certain tasks didn't “fit” into the system. There was general agreement that there were actual work tasks such that the format in Accord couldn't contain them and there was a discussion about how categories could be added
to account for them. The matter was left at that, but, interestingly, a discussion at a Lambda team technical leaders meeting one week afterwards (late November 2007) tied in with this issue. They were discussing peoples roles and responsibilities on the Lambda team, especially in relation to the Scrum methodology they were employing. What is interesting was how this tied into the issue of tasks in Accord, because they also had to consider what roles were defined in the system and how they might have to either adapt them or adapt to them.

Two days after this technical leaders meeting, the Lambda team had a lessons learned meeting as a development sprint was concluded. Some developers again raised the issue of Accord not accommodating their particular work well. One developer in particular said that she was “still not comfortable using it” (Transcribed notes 2007-11-30), since her job involved “little tasks” and Accord was not good at accounting for this. After the meeting, I interviewed that developer about her problems. Her job was to provide support on the ERP system that the product utilized within the entire Lambda project. So people from all the various sites would contact her – she was a member of the Lambda development team in the Dublin office - when they had trouble with this system or questions about how things worked with it. This meant, she told me, that her job was about having conversations with people asking for help, trying to understand the problem first and then helping them work out a solution. She described it as “more knowledge transfer than coding” (Transcribed notes, 2007-11-30) and told me that most of the stuff was done in about 20 minutes. The problem she was having was that there was no good way of representing this kind of task in Accord and if she were to enter every single task, she'd be spending more time doing that than actually solving the problems.

Following up on this issue, the Lambda teams technical leaders had a discussion at their weekly meeting in early December 2007 where they talked about what could be “tracked” in Accord. After the meeting, I interviewed the project manager about these discussions and she told me that the “role and responsibility issue” - as they were referring to it - actually stemmed in the fact that they were discovering how peoples actual roles differed from the the formal roles defined for the team within the project. But, she said, this had become more acute since the introduction of Accord, both because peoples roles had changed with the introduction of that and since they now had
to adapt to the roles that were "written into" the system itself.

What we have seen here is how there was a need to co-adapt the roles of the team and those defined in the new system. But, importantly, it also illustrates how people in different roles had different experiences of the introduction of the Accord system.

5.4.5 Contingencies staggering the work flow of the Lambda team

On an afternoon in early December 2007, just as the work for the day was beginning to wind down, the project manager discovered that she couldn't access the Accord server. She went to find the developer responsible and he confirmed that the server was indeed down, the reason being that he was updating the system. The project manager passed on the information to the development manager, but since the work day was at an end, the matter was left at that. However, on the following morning, the Accord server was still down. When I discovered this on that morning, I went to speak to the developer responsible and he confirmed that they had run into some problems with the update process and that it had not completed properly. As the team leaders began coming in, they gathered around for an impromptu discussion about the situation. The main point made was that, while it was definitely a difficult task to update the system, it shouldn't have been attempted during live development. But the question was left at that and they let the developer responsible work on solving the problem. One of the technical leaders joined in to help him. While the developer investigated the source of the failure by reading available documentation and asking in online forums, the technical leader was looking for an available machine they could test the update process on. The developer was also talking a lot to other developers on the team, either just articulating the problem or asking them for bits and pieces of information.

When the team leaders met later that afternoon, the update issue was the main topic on the agenda. It was explained that they had wanted to update the system because of a new function, but since the update had failed they decided it was better to roll back to the previous working version. Impact on work flow could be minimized, it was argued, and the responsible developer could spend some time learning how the update could be done properly. The technical leader who had been helping the developer earlier that morning also told them about their idea to find a separate test machine where they could run tests before attempting to update the live server. It was also argued that the main
cause of the error was inadequate or poor documentation on how to actually do it. Therefore, the team architect suggested that they needed to begin building up their own “knowledge-base and documentation” about such procedures.

Following up on these events, I interviewed the developer responsible a few days later, asking him about his view on what had happened. He told me that the main reason for the failure was lack of proper documentation about migration procedures, something related to the fact that Accord was still only in development itself. Interestingly, I had reflected on the fact that it had seemed like the ones mostly affected by the downtime were the team leaders and managers. When asked about this, the developers comment was that task tracking was more “about process than development” (Transcribed notes, 2007-12-12). He continued to say that he didn't consider it an issue. Had he been given more time, he would have completed the update. But people, he said, got upset when it was down and therefore he had to halt the update work and roll back to the old version to keep it up and running.

This is a specific example of how the contingencies introduced with Accord caused problems, but it also illustrates further how there was a learning process. In this case, they encountered an issue related to this novel contingency. The breakdown caused them to reflect on the situation and thus prepare for future issues of similar nature.

### 5.4.6 Summary and discussion

This section has presented an episode in which a novel collaboration platform – given the pseudonym “Accord” - was introduced to the Lambda project. The vignette centred on how the Lambda team utilized this novel platform in their work. (See Figure 7 on page 99 for an overview of the sequence of events depicted.) It again exemplifies the topic of evolving technology (introduced in 2.5.2.1), particularly in terms of the intricate ties between technological change and changes in organisation and practice.

What we have seen in this vignette was a process where the Lambda team integrated the Accord platforms structuring properties within the particular project setting. In this process, there were many problems related to the contingencies that the introduction of Accord brought with it, stemming from the fact that the platform itself was still under development. This meant that it was necessary for ongoing negotiations within the Lambda team about to what extent they were to rely the systems services. These
Exploring issues of change in technology, participants, and goals

Chapter 5

Contingencies are an important aspect and one that I will elaborate on further in chapter 6 when discussing the common themes between the different episodes.

Furthermore, it was also clearly shown how change of technology was intricately tied to the evolution of practice, as the Lambda team purposefully adapted their practice in relation to the new platform. This was not a matter of instant change of practice. Rather, there was a process of appropriation as the Lambda team gradually learned to use the system consistently within the group. Problems accumulated because people based their usage on previous experience with similar platforms and therefore had slightly different assumptions and practices. These problems were discovered only as collaboration took place and breakdowns occurred. From a practice-based perspective on knowledge (outlined in section 2.3), learning to use the new platform as part of the collaborative effort was a social and participative activity rather than merely a cognitive one.

Accord became the platform for collaborative software development within the whole Lambda project and decisions about its introduction had to be taken in consultation with all project stakeholders, but it was the Lambda development team who led the introduction. They installed and maintained it. However, even though Accord mediated the interaction between all stakeholders in the project, and all the teams involved had to utilize it into their work, there was no significant discussion in the Lambda team on how the practical issues that arose when they began using the new platform might also impact remote teams. The impact of contingencies were considered for the whole project, but the learning process I observed in the Lambda team was confined to that site. So while the platforms structuring properties had an impact on the project as a whole, putting it into practice was treated as a challenge for the individual team. This particular aspect of the episode provides a concrete example of the situated nature of knowledge in distributed settings, confirming a potential issue for knowledge sharing on a project level (as compared to a practice level, which I am concerned with) that has been highlighted in the literature (Sole and Edmondson 2002; Nicholson and Sahay 2004; see also section 2.2 for a literature review on these issues). I want to flag this particular finding, as I will return to it in my discussion of how the purposeful adaptation of practice may change our understanding of what situated means in chapter 7.
5.5 Vignette 3: Participation and enculturation in the PyPy open source community

Let us now turn attention to the second study that is included in this thesis, the case study of the PyPy open source community. The first vignette from this case is an account of my own experience of participating in the community (see section 4.3.2 for more on how this was part of the research process). It exemplifies how issues related to enculturation and socialisation (introduced in section 2.5.2.2) were dealt within the particular context of the PyPy community.

The vignette focuses on the process of me joining the community and attending my first sprint. However, I also complement my account with excerpts from a broader analysis of the community's mailing lists and the results of the email questionnaire that we distributed to project participants. By doing this, my experience is put in relation to the observable experience of others who have joined the community.

Figure 8 on below contains an overview of the main events depicted in this vignette, and the reader is also recommended to consult section 4.3.1 for some basic information about the PyPy community.

Figure 8: Sequence of events in vignette 3.
5.5.1 A newcomer joining the community

This part of our study of PyPy took place in February and March 2007. The goal from the outset was to perform participant observation of a sprint. Particularly, we aimed at a sprint what was to take place in Hildesheim, Germany, in early March 2007. I began interacting with the community roughly a month prior to this event. The first steps were taken online, subscribing to the mailing list and announcing my interest to the community. Because my participation was part of a research effort, I began by contacting the core members of the community. They approved of the idea and immediately gave me advice on what parts of the project’s documentation would be a suitable starting point. What was suggested as relevant included both technical and architectural documentation, and documentation about work practices and customs within the project (all available online\(^{28}\)). Interestingly, this was suggested in terms of how to best prepare me for the upcoming sprint, something we shall return to below. In addition, I was also encouraged to ask questions and begin participating in the available online forums. Once this was done, I made an announcement in the public mailing list, explaining my dual role as a participant and researcher, and writing briefly about my technical background and current interests (relating to the project at hand).

Since I had already been discussing my intentions with the core members and they had given advice, there was no immediate response to my mail, apart from a later exchange regarding practicalities such as travel and accommodation. My experience was in this respect slightly different compared to what normally happens when newcomers approach the community and ask for advice on how to get started. It contained the same elements, but the route was different, as I went directly to the core members first before addressing the wider community. But if we look at a few examples from the mailing lists of others approaching the community, we see the same focus on looking up relevant documentation:

1. Cool! Contributions are of course very welcome! I guess the most immediate step would be to read through the documentation and ask any question you might have (here – on the mailing list or on the IRC channel). It certainly won’t be a problem finding work for you :-)

2. Please look at the getting started part of [http://codespeak.net/pypy](http://codespeak.net/pypy) for recommendations regarding subversion and other aspects of pypy development. There is also extensive documentation to read if you are interested :-)\(^{28}\)

\(^{28}\) [http://codespeak.net/pypy/dist/pypy/doc/](http://codespeak.net/pypy/dist/pypy/doc/) (Accessed 2010-02-08)
After brief introductions of backgrounds and interests, others suggest relevant online documentation to look into. As we can see in the quotes above, there is also the immediate invitation to ask questions and begin discussing intentions at the available online forums. Additionally, there's the same emphasis on these initial activities as preparation for eventually attending a sprint:

(1) Could you please tell us a bit about your expectations and interest in PyPy? It will help us before and during the sprint to know...See you in Dallas!

(2) This sounds very good. I think it's ok if you arrive wednesday evening and join us in some pub. We should make sure though, *before* the sprint, that your account and dev environment works and you find your way to the source and all. Of some help might be the documents that you find in the 'doc' section left hand at

http://codespeak.net/pypy/index.cgi?doc

> I look forward your response and to meet you all in person soon.

I look forward to meeting you, too, and think that your background will allow you to participate in all areas of PyPy. If you like it would be great if you can come up with some questions on pypy-dev that we would answer publically. That helps you and probably many others as well as us to force us to clarify things

Thus, reading the technical and methodological documentation, familiarizing with the source code, and beginning to discuss ideas and intentions in online forums were considered to be the essential introductory steps for people joining the PyPy community. In my experience, what this provided was an insight into the technological architecture and how to go about contributing to it, but, equally important, also about who was who in the community. This, as we shall see next, played a vital role when attending ones first sprint.

5.5.2 The newcomer attending his first sprint

Once the first steps had been taken online and I had spent some time working with the project on my own – guided by the advice given when approaching the community – it was time to travel to the sprint. Prior to the event, the practicalities of travel and accommodation were coordinated on the specially dedicated sprint mailing list (pypy-sprint). The particular sprint attended took place in Hildesheim, Germany, in March 2007. Interestingly, this was actually the site of the very first PyPy sprint, where the original – and for the most part still active and central – members of the community had
met to brainstorm and make the founding design decisions for what would become the PyPy project (section 4.3.1.2 contains a brief history). This time was perhaps not of equal historical importance, but it was still a special sprint, as part of its purpose was to finish up reports due for the completion of EU funding the following month. It was organized in two parts. The core members of the community had gathered for 3 days to collaboratively finish writing these reports, and this was followed by 3 days of development work open to any participant. It was for the second half of the Hildesheim sprint that I attended.

Well at the sprint, the newcomers – there was myself and another person at this particular sprint – were effectively taken by the hand. Tutorials were organized in order to help us get started in our work. This is common practice for PyPy sprints, as we can see in these examples of sprint announcements:

(1) Sunday: Starting at 10:00. This day is focused on getting to know PyPy enough to start to participate. We will hold a PyPy tutorial and an architectural overview. Planning meeting for the work to be done during the week and grouping of developers (pairs or groups mixing new participants with core developers).  

(2) The first day of the sprint will be a tutorial day, were talks about PyPy, Squeak and sprint-driven-development will be given. The furtherdays will be sprinting days.

Sprints in the PyPy community begin each day with a status meeting. On the very first day, this meeting takes the form of a “start-up meeting”. The participants introduce themselves – if necessary - and discuss their intentions for the next few days. During this discussion, the rough goals outlined prior to the sprint are adjusted according to the participants wishes and personal goals. The eXtreme Programming practice of pair programming is central to PyPy's sprint, so the next key part of this meeting is deciding who are going to pair up and what they are going to work on according to needs and wishes. During the rest of the sprint, the status meetings are for reporting progress and directing the groups efforts, including changing of pairs if there is a need or a wish. All major decisions are taken by the group as a whole during these meetings, there is no single individual who coaches the effort (in contrast to sprint practices in many other communities, where there is usually a designated coach for the whole event).

29 http://codespeak.net/pypy/extradoc/sprintinfo/tokyo/sprint-announcement.html (Accessed 2010-02-09)
If there are newcomers present the initial status meeting is also used to discuss topics for tutorials based on their interests and previous knowledge. In our present case, the core members had formed an idea about what would be suitable topics after the introductions made online, but an equal emphasis was placed on what we felt we would get the most out of. Following a brief discussion at the start-up meeting, the particular topics were decided upon. Who was going to hold the tutorial depended then on the tutorial topics and which of the core members with relevant expertise were present. One of the original project founders took responsibility for our tutorial, which was aimed at giving an overview of the architecture. In true bricolage spirit, he used slides from a talk he had previously given at several conferences. Interestingly, in the spur of the moment, two other core members decided to sit in for the tutorial. They overheard what the topic was going to be when it was discussed beforehand and said they wanted to join in for a while, that maybe they could be of assistance. During the tutorial they intervened and commented regularly on what was being said, offering additional information and explanations, but also sometimes debating the points being made or discussing the way certain things were described.

Once the tutorial was finished, we were asked if there were any particular areas we wished to work on. Pair programming was a central practice in PyPy sprints and the core members who had given the tutorial suggested that we pair up with two more experienced developers. They currently didn't have a particular task to work on and were interested in working on a part of the system which was also deemed suitable for someone trying to get to grips with the inner workings of PyPy. The rest of the first day – the tutorial had finished around noon – was spent working with those developers. Putting this experience into perspective, it has to be said that newcomers getting to team up with experienced developers was an explicit strategy in the PyPy community. In addition, it was also considered good practice to switch pairs as work progresses:

1. Planning meeting for the work to be done during the week and grouping of developers (pairs or groups mixing new participants with core developers).\(^{31}\)

2. Change teams as often as possible especially for first-time pypy-sprinters so they get to work with everyone.\(^{32}\)

\(^{31}\) [http://codespeak.net/py/pypy/extradoc/sprintinfo/tokyo/sprint-announcement.html](http://codespeak.net/py/pypy/extradoc/sprintinfo/tokyo/sprint-announcement.html) (Accessed 2010-02-09)

And, in line with this, I got to pair up with another experienced developer to work on another part of the system following the scrum meeting of the second morning. Similarly, on the third day, I got to pair up with two new developers (small groups of 3-4 developers were also common). This switching of pairs meant that the newcomers got to see the PyPy system from several different perspectives, both high-level and low-level, depending on what parts they were working on. But it also meant that they got to meet and work with several experienced developers. In addition, social activities were organized during the sprint. Lunches and coffee breaks were regular elements of the working days and on the first evening a dinner was arranged at a restaurant in town. Being at the sprint wasn't beneficial for a newcomer solely because of the canonical practice put in place, but also because it was a chance to get to know the people who you had interacted with online beforehand. The importance of getting to know people face-to-face was also emphasised by the respondents of the questionnaire:

(1) I chose to go to the Duesseldorf sprint in May-June 2006 because I was invited by the other developers; I wanted to do more work for my thesis and also to meet in the real life all the other people I was used to talk with by IRC.

(2) I wanted to meet all those people that I had only interacted online with to that point. Also, I wanted to pair-program with some people, which is rather hard to do via the web (but possible).

The explicit emphasis on accommodating newcomers at sprints was apparent during the observations of the Hildesheim sprint. Other examples of how sprints were tailored to accommodate newcomers are plentiful as well. For instance, at the other observed sprint – in Limerick, August 2006 - the schedule was adapted to accommodate two local developers who were joining late in the week and this has happened in other sprints as well:

Originally, we intended to go rather directly for translation and thus for a first release of PyPy. But before the sprint we decided to go differently about the sprint not only because Michael Hudson and Christian Tismer had to cancel their participation but we also wanted to give a smooth introduction for the new developers attending the sprint.33

What we have seen here is how the efforts of helping newcomers at sprints were a continuation of the online introductions made earlier. Tutorial topics, work tasks, and pairs were decided based on previous knowledge, interest, and what developers were

33 http://codespeak.net/pypy/extradoc/sprintinfo/AmsterdamReport.txt (Accessed 2010-02-09)
Important was also – just as in the initial activities online – how efforts were aimed at providing both help with actual technical work tasks and opportunities for getting to know one's colleagues in the community.

5.5.3 The newcomer meeting the gurus – and their expectations?

PyPy is a very complex technological project. Its founding members are highly regarded in open source circles and are very much the guru's of the community, having both the most intricate knowledge about the product and the best overview of the different development streams of the project. As work carried on at the sprint I attended, one noticeable activity was for these guru's to “wander”. They would simply spend a lot of time walking around the locale as people were working. Sometimes they would stop and ask someone about what they were doing or offer some advice, often they would get called over as they passed by. Wandering was a way of making themselves available. Through this behaviour they would remind people of their presence and show that they were available to answer questions. While mentoring practices were obviously central at PyPy sprints - as seen in how newcomers were given tutorials and got to pair up with more experienced developers – the importance of this kind of implicit or informal mentoring must not be diminished. For a newcomer, it provided very valuable support, but also encouragement through the fact that they took an interest in everyone's work! It also served as a mechanism for these core members to keep themselves informed about all the different threads of work being carried out at the sprint.

As was mentioned earlier, not everyone approaching the PyPy community online would be invited to join a sprint. There was an unofficial screening process whereby it was the people who displayed skill, enthusiasm, and a willingness to contribute significantly who were asked to join sprints. While the official attitude was that anyone who knew the Python programming language and had read the PyPy documentation was welcome and able to contribute, I soon noticed how my level of skill was different from what was implicitly expected. As collaborated with the other developers, a large amount of time came to be spent explaining core concepts and ways of thinking. While I did know Python and had a fair overview of the PyPy architecture, I lacked any experience with implementing programming languages. This also meant that I had to get a lot of help regarding what to work on, as it was difficult for me to grasp what areas were suitable
for development. While they did receive plenty of guidance, the newcomers were implicitly expected to be able to take initiative as to what to work on and what implementations that would work. The guidance rather aimed at directing their effort with respects to the design philosophy and current goals of the PyPy project.

5.5.4 Summary and discussion
This vignette has illustrated how a newcomer joined the PyPy community and attended a first sprint. It was an account of my own experience of participating in the PyPy community complemented with excerpts from an analysis of mailing lists, online documentation, and open email questionnaire (see 4.3.2 for more on how these different methods were utilized in the study of PyPy). (See Figure 8 on page 108 for an overview of the sequence of events depicted.) The episode exemplified issues of enculturation and socialisation (see 2.5.2.2 for an introduction of this topic) in how the PyPy community introduced new members into their community of practice.

First of all, what emerges as particularly important in the above account, is the interweaving of online activities and sprint attendance. The initial steps in the online forums were very much in preparation for the first sprint, which unfolded in relation to the prior experiences online. In doing so, the PyPy community have come to utilize the strengths of both mediums. Online, the newcomer gains a good overview of the project and the community because of the availability of general documentation and the ability to ask questions directed towards the whole community. The sprints, on the other hand, facilitate more in-depth understanding of both technology and norms, as the newcomer gets a guided first-hand experience of intricate development issues and a chance to interact face-to-face with many of the core members of the community. Thus, practices for receiving newcomers in PyPy had evolved a focus on enabling both the technological competency and the community membership necessary to become an independent contributor to the project. Echoing the words of Brown, Collins et al. (1989), becoming a participant in the PyPy community was learning to be a competent practitioner rather than simply learning about the project.

Another important aspect of the episode, is the free-flowing nature of the interactions and decisions. Nothing was predefined, and decisions were taken entirely based on particular circumstances, including newcomers wishes and prior knowledge as well as
the PyPy project setting and available expertise. For instance, who was going to lead the tutorials at sprints depended on what the newcomers considered would be the most beneficial topics and who with expertise relevant to that were available to give them. This is important because it meant that, while there were standard practices, they were actually facilitating the tailoring of introduction for individual newcomers.

Both of these findings are noteworthy for my argumentation about the purposeful adaptation of practice, which will be further elaborated upon in chapter 7. The theme of continuous change – as seen in how the PyPy community had practices for dealing with a constant flow of individual newcomers – is elaborated further on in my discussion of commonalities between the different episodes in chapter 6.

### 5.6 Vignette 4: The PyPy community negotiating project goals and external expectations

In this vignette we look at how the PyPy community dealt with ambivalence regarding the future direction of the project following a major release. In particular, we shall follow a extensive discussion that took place in the community's main mailing list. This exemplifies issues of emerging project purposes and goals, which were introduced in section 2.5.2.3.

Figure 9 on page 117 contains an overview of the main events depicted in this vignette, and the reader is also recommended to consult 4.3.1 for some basic information about the PyPy community.

#### 5.6.1 PyPy deciding what development areas to prioritize

PyPy-dev is a public mailing list, so anyone can subscribe to it or access the archives. Together with the IRC chat channels, this mailing list makes up the community's main discussion space. The episode at hand took place just after the release of the 1.0 version of PyPy. Following this release, the community's weekly “sync” meeting in the IRC chat channel had been dedicated to discussing the future direction of the project, particularly in terms of “to-do” issues. The discussion thread we are going to follow was initially titled “Work Plan for PyPy” (it was a branching discussion that eventually resulted in many different threads) followed up on the discussion from the meeting. It

was initiated by one of the founding members, who summarized the agenda and main discussion points from the preceding meeting:

Here is a summary of what we discussed today on #pypy-sync. We tried to list the tasks that now lie ahead of us. There are a number of relatively independent tasks, but there is also a knot of dependent tasks that we'll have to untie bit by bit.

Following this, he listed a number of particular tasks and also stipulated how they were related to each other. There was a brief exchange of comments on the task list and people also suggested some additional points. But in this exchange there was also a side-note which became a branch of the discussions. One of the contributors mentioned a question that was asked in the chat meeting regarding a particular part of the system:

Finally, a note which is not related with the work plan but to a question that appeared in the logs; someone asked whether we want to "sell" RPython as a stand-alone product.

He continued by telling people of a talk he had given about PyPy and how the audience had seemed to perceive this part of the system as the most relevant outcome of the project. He concluded by asking:

It would be interesting to know if this perception is world-wide or only limited to the
attendants of my talk (maybe because I didn't stress enough the pros of the other pypy goals or the cons of rpython).

A peripheral follower of the project replied and confirmed that he also found this part of the system to be useful, but most of the core members maintained that this was not the most fruitful part. Another core member entered the discussion at this point and told a similar story of a talk he had given. His audience had emphasised another, broader, aspect of the system. He went on to suggest that if they could develop this aspect to a more usable state, users would also be free to explore the part that had earlier been emphasised, if they so choose. However, in the response to this, it was still maintained that priority should be put on other parts of the system:

From my POV [Point Of View] maintaining all of parts required to use RPython as a general purpose language is a bit overkill, and people are really interested in parts which they yet don't know they're interested in.

At this point, another peripheral follower of the PyPy project wrote a long entry in the debate, criticizing the expressed opinions and making some radical suggestions. This person, while only a peripheral follower of the PyPy project, had significant experience and celebrity in the open source domain, particularly within the larger Python community. Above all, his argument was that people outside the PyPy project had a different perspective, but that it was vital for the PyPy community to recognize the value of their point of view when making decisions about future directions:

An open source projects needs mindshare. It needs users. PyPy is an open source project, and I'd recommend getting any users you can get. Granted, you'll get opportunists like me who are only there for production benefits. These people are less interested in language experimentation and new features - they're there for the speed primarily: the main thing that PyPy, through RPython, can offer to a Python programmer today.

He was directly addressing several of the people who had posted opinions previously and was, in fact, making quite direct propositions of what they ought to do. The argument was that if the PyPy community would emphasise the part of the system that people had expressed an interest in, it would follow that people interested in contributing specifically to that part would be attracted to join the project. The problem of few developers currently being interested in working on that part would, one could say, solve itself.
Chapter 5 • Exploring issues of change in technology, participants, and goals

5.6.2 PyPy's initial response to external expectations

The immediate response to this entry came from yet another peripheral follower of the PyPy project, who agreed with the points made and suggested that perhaps they should branch the project. However, one of the core members – and a leading voices in this discussion that far - also replied soon after, still maintaining his standpoint that this was not an area they would want to go into to much. The reason, he argued, was related to priorities and how people involved in the project wanted to spend their time. Since, he said, this was not an area that any of the core members were particularly interested in, they shouldn't emphasize it, although if some new contributor wanted to they could of course.

A day passed and the general discussion continued on some specific technical issues that were mentioned in the original email. At the same time, the core community members were also discussing these issues in their live chat channel (IRC). The next entry of interest in the email list discussion was an answer partly to what had been said in the chat rooms and partly to what was being said in the list. However, at that point, another core member brought the discussion back to the issues highlighted by the external observer:

*The reason why this stuff kept me busy thinking since the meeting was the fact that we are about to trash exactly the little stuff people are actually using. We do have very little users, and most of them are involved right here.*

*So if we are going to trasg stuff, then we should go and offer help to those few users, to not getting distracted from PyPy.*<please assume that I could insert all of Martijn's text, too>

Two days passed, at which point the external observer hooks into the discussion again, reiterating his points. He addresses the developer community directly, though he toned down his aggressiveness somewhat:

*Please note that I'm not actually *asking* for anything but information. I'm not telling the PyPy developers they should make commitments and do various kinds of work. That's up to you. I'm just trying to guess what other developers might want to know.*

*I would urge the PyPy developers to consider investing in the open source community by supporting features like this and promoting them. This would mean serving admittedly self-interested developers with short term goals. The promise of immediate payoff is a great way to attract people to an open source project. If there is the hope of continuing payoff (and PyPy has tremendous potential here), some of those new people can be expected to become*
valuable contributors to the project as a whole. And yes, I'm self-interested in saying all of this - that's my point.

This entry, though, was met only with silence.

5.6.3 PyPy deferring the discussion and outsiders perceiving it as “answer by silence”

After this entry, two days passed without there being any response, after which the external observer re-iterated his thoughts:

I posted this to a longer thread the other day but have received no answers to it whatsoever. Perhaps it got lost, so I'll do a repost. I spent a bit of time writing this post, and I ask some questions in it that I think should be answered. Perhaps by telling me I don't understand what I'm talking about or that my analysis is entirely incorrect. Perhaps it was simply too long?

Whatever it is, I do think the PyPy community needs to be aware of how technical decisions have an impact on attracting or repelling potential contributors. "answering by silence", which happened so far, is one way to make potential contributors worried. :)

He was putting some pressure on the PyPy community and a core member with a facilitatory role (related to the EU funding) stepped in to provide an answer. Interestingly, however, it was not a direct answer to the questions asked, but rather an attempt to soothe the situation:

Although I can't answer your questions I do appreciate the time and effort you and others have spent on this issue during the last couple of days, and I am sure the PyPy core gang appreciates your input.

In addition, she also made the point that they would be meeting at a then upcoming sprint to discuss these issues and suggested that this was probably the reason for why there were no direct answers at that point:

The current plan is to meet at EuroPython in Vilnius to have strategic technical discussions regarding direction and among them decisions regarding the issues you raise in this email. I interpret the silence as people feeling that they can't answer this until such a discussion (???) but that does not excuse the total silence - hence this email (that does not give you the answers either).

In response to this, the external observer was understanding. It was very much in accordance with the open source attitude and his answer reflected this:
An answer of "we're thinking about it, we can't answer it yet" is totally acceptable. Thanks!

Other core members of the community also added to this line of reasoning, confirming how the particular topic of discussion was deferred until they could meet face-to-face at the next sprint:

[1] That's a very good list of questions. Indeed, we don't have clear answers for them yet, but just vague thoughts. I think it's fair to say that we'll discuss this at EuroPython and make some commitments then; for now we're only tossing ideas around.

[2] I think that discussing things live at the EP makes a lot of sense :)

Also I think that several points regarding what is going to happen needs clarification and so on.

Let's talk about it live.

Discussions about particular technological solutions and strands continued for a few days, then the sprint took place. After the sprint, there was a gap before this particular discussion thread continued. Although there was no direct answer to the questions posed, the external contributor was happy with the progression:

I'm glad to see this important discussion is taking place again. For my input, see my mails from a few months ago. :) [19 September 2007]

Even though the issues were far from resolved and discussions continued from that point, we are going to leave this episode here. What has been illustrated is how the PyPy community were negotiating their path forward and how this was influenced by peripheral followers of the project. We also saw how the availability of sprints provided a forums to which controversial discussions could be deferred.

5.6.4 Summary and discussion

In this last vignette, we have seen an episode where the PyPy community were negotiating the direction of the project following a major release. (See Figure 9 on page 117 for an overview of the sequence of events depicted in the vignette.) It has exemplified the topic of emerging project purposes and goals (introduced in section section 2.5.2.3 ) within the particular context of the PyPy community.

Open source communities commonly have emerging purposes, as the goals of the project are in constant negotiation within the changeable community. This vignette portrays an episode where this was particularly concrete in the form of dedicated
discussions following a major release. We also saw how this discussion involved the PyPy core members responding to peripheral followers expectations about the projects future direction. As the episode unfolded, the PyPy core members came to defer the public discussion until after a then upcoming sprint. The benefits of using the collocated events for complex or controversial discussions was something that the PyPy community had learnt over time (shown and commented on in section 4.3.3.4). But it was also important that this be made explicit. “Answering by silence”, as was pointed out by one peripheral follower (p. 120), can make potential contributors worried. In the exchange we saw, the peripheral followers of the project who joined the discussion were clearly worried about whether the silence should be interpreted as if the community was adrift on these issues. Thus, even though the PyPy community members meant the deferring as a way of reducing uncertainty, there was also a potential danger of the opposite.

Considering external expectations on the project is an important theme which I will elaborate on in chapter 6, while the practice of deferring complex and controversial discussion plays an important role in my argument about the purposeful adaptation of practice in chapter 7.

5.7 Summary of chapter findings

In this chapter I have presented four vignettes that exemplified three topics related to change within the particular settings under study in this thesis, namely evolving technology, enculturation and socialisation, and emerging project purposes and goals. It was a continuation of the empirical analysis initiated in chapter 4. There, practice in the two settings under study was elucidated, whereas this chapter has examined in more detail these particular topics. The topics were in turn suggested in chapter 2, where I argued that they are particularly relevant topics for investigating how changing circumstances is dealt with in distributed software development practice (see particularly section 2.5.2).

The first vignette exemplified the topic of evolving technology, illustrating an episode where the Lambda team had to deal with a technological contingency and adapt their product following an update in the software framework it built on. The second vignette provided another exemplification of the topic of evolving technology, particularly in
terms of the intricate ties between technological change and changes in organisation and practice, when it illustrated an episode when the Lambda team utilized a novel collaboration platform that was introduced to the project. The third vignette illustrated an episode where a newcomer joined the PyPy open source community and attended a first sprint, exemplifying the topic of enculturation and socialisation. And lastly, the fourth vignette exemplified the topic of emerging project purposes and goals by illustrating an episode where the PyPy community were negotiating the future direction of the project following a major release.

The topics and main findings of each individual vignette are summarised in Table 5 above. In relation to the topic of evolving technology, I have shown that, in the settings under study, it involves contingencies that extend beyond the project itself, and how there is a complex interplay between actors both within and outside the local groups following technological change. I have also illustrated how there is reflection in preparation for future situations of a similar nature and purposeful adaptation of practice.

<table>
<thead>
<tr>
<th>Vignette</th>
<th>Topic</th>
<th>Findings</th>
</tr>
</thead>
</table>
| Lambda development team dealing with technological contingencies | Evolving technology          | · Complex interplay between local and remote actors in negotiating a technological contingency  
· Subsequent reflection on how to deal with issues of software updates |
| The introduction of a novel collaboration platform in the Lambda project | Evolving technology          | · A number of contingencies were introduced together with the new platform  
· Learning to use the new system consistently within the group was key issue |
| Participation and enculturation in the PyPy open source community | Enculturation and socialisation | · Interweaving of online activities and collocated events facilitated enculturation  
· Practices for tailoring introduction to individual newcomers |
| The PyPy community negotiating project goals and external expectations | Emerging project purposes and goals | · Considering external expectations when deciding on priorities  
· In this particular instance the intentions had to be communicated more clearly |

Table 5: Summary of findings from the four vignettes.
Chapter 5 • Exploring issues of change in technology, participants, and goals

by gradual adjustment. For enculturation and socialisation, I have shown that interweaving of collocated events was important for achieving both technological competency and community membership, a practice that had evolved in the PyPy community in response to the constant flow of individual newcomers. I have also shown how this practice specifically provided a tailored introduction for individual newcomers. Lastly, in relation to the topic of emerging project purposes and goals, I have illustrated how this involves negotiating external expectations, something that the PyPy tackled through their practice of deferring complex and controversial discussions from online forums to sprints.

Having delved into the details of these particular topics in the settings under study, this chapter has provided additional empirical evidence for achieving my objective with this thesis. There are findings related directly to my argumentation about the purposeful adaptation of practice in relation to changing circumstances, which will be utilized in chapter 7. Furthermore, the findings here provide the necessary empirical material for making a characterisation of how changing circumstances unfolded in the settings under study. I will utilize this in the following chapter, where I discuss what the common themes are between the different episodes.
6  Characterising how change in circumstances unfolds

6.1 Introduction
In this chapter, I discuss what shared themes emerged in the four vignettes in the previous chapter (number 5), which illustrated episodes of evolving technology, enculturation and socialisation, and emerging project purposes and goals within the Lambda team and the PyPy community. The contribution towards the objective of my thesis is a characterisation of how episodes of changing circumstances unfold based on my empirical analysis. This particular aspect of the software developers work is of interest for understanding how they develop their practical knowledge in dealing with changing circumstances. In the following chapter (chapter 7), I will argue that this is achieved through the purposeful adaptation of practice. Utilizing the insights from my discussion here, I will be able to suggest what the main concerns of this process of adaptation are.

Three themes are highlighted throughout the chapter. Continuous change is the first. My argument is that, in neither vignette was the problem one of dealing with distinct or discrete change, but rather one of dealing with ongoing and intertwined change as factors that kept changing and had to be dealt with in parallel with numerous other issues and activities. Equivocal situations is the second theme, referring to how the software developer groups had to decide on courses of action in situations that were not clearly or precisely determinable for the actors. External influence is the third and last theme. With this I refer to how all the episodes involved relationships – primarily professional and technological – extending beyond the projects themselves. I will concretely show how these themes pervaded the vignettes and discuss the implications of them for our understanding of the complexities of distributed software development.
6.2 Continuous change

The four vignettes presented in chapter 5 all illustrated episodes where the circumstances were changing, particularly in terms of technology, participants, and goals. The first theme that I want to discuss regards the fact that what was illustrated was, not distinct and discrete, but ongoing and intertwined change.

While the Lambda team (see section 4.2.1 for an introduction of the setting and section 5.3 for the vignette) were developing their own product, the software framework it built on was also autonomously under development. The first vignette spoke of how the team were finishing a new patch at a time when there was also a beta version of this framework in circulation. This meant that they had to decide whether to integrate support for this new version or not. As it happened, they decided against it and continued working on their patch. However, some time later, only days before the patch was expected, the circumstances changed. The new framework version was finally released. However, considering carefully the pros and cons, the team corroborated their earlier decision and carried on without supporting the new framework version. But the circumstances changed yet again when the decision was reversed by a remote manager. This imposed on the team an urgent need to implement support for the new framework version, illustrating the necessity for the team to be prepared for ongoing change often beyond their control.

The introduction of Accord, as described in the second vignette (section 5.4), did of course mean a major change for the Lambda team. As a wide-ranging collaboration platform, it constituted the tool set for everyday development work in the team. However, it must be pointed out that although the vignette began with describing how the team was informed about Accord at a team meeting, it's meaningless to say that this was the point of change. Accord had been discussed for a long time before that and, as we saw in the vignette, it was by no means a clear-cut introduction. The extent to which they were going to use the services of Accord was subject to ongoing change. While task tracking and project status services were utilized early on, the team realized that there was a set of issues that needed to be considered before making use of the source code management services, particularly because Accord itself was still under development. Due to these issues, switching source code management systems never took place, but instead became a subject of ongoing debate and uncertainty (another
overarching theme that I will elaborate on in the following section). Additionally, the teams relationship to the new platform was also constantly changing as emerging issues were raised by participants. For example, the team leaders realized that problems were accumulating due to people having different ways of using the system, and this inspired a new focus on developing shared practices. Another concern was how the introduction of Accord dissimilarly affected people with different roles, which sparked a discussion about how to co-adapt the roles of the systems and the roles of the team. Lastly, we also how the continued development of the Accord system itself gave rise to problematic situations, particularly in the form of update timing and procedures.

Turning to the PyPy open source community (refer to section 4.3.1 for an introduction of the setting), the same theme of dealing with ongoing change emerges. In the vignette describing how a newcomer joined the community (section 5.5), we can see this from two points of view. For a newcomer, beginning to participate and eventually attending a sprint is all about change. He or she begins working on a new technical project, but it also involves collaborating with new colleagues and learning the practices of how to contribute in this particular community. Much of the change is related to the particularities of the PyPy community as compared to the newcomers prior experiences. But the reception and introduction of newcomers is also about dealing with change from a community point of view. The practices for receiving newcomers in the PyPy community had evolved to deal with the constant arrival of new participants. Additionally, it is evident from the vignette how the process of beginning to participate itself can be characterised as a process of consecutive change. A significant example relates to the interweaving of online and offline interactions, as the newcomer moves from having interacted with colleagues online to meeting them and working face-to-face at the sprint. Another was the pair-programming practice at the sprint itself, where the newcomer gets to pair up with different developers each day to work on different parts of the system.

Lastly, the fourth vignette (section 5.6) described a situation where the PyPy community were negotiating the future direction of the project following a major release. Some major development goals had been achieved and participants were debating what areas to explore in the future. The mailing list discussion we saw in the vignette was one of the main outlets for these negotiations. Thus, what we observed was
change on a broader scale, as it was the projects general goals that were being updated. But it has to be acknowledged that this was not a unique event. In an open source project such as PyPy, the projects purpose and goals are continuously evolving and this was just an episode where such discussions were particularly prominent. Interestingly, the account also shows us how the focus of the discussions themselves kept changing, particularly in connection with peripheral followers of the community joining in and offering opinions on where the project should go. As the situations changed with added external pressure (which is also another overarching theme that I will elaborate on further in section 6.4 below), the discussions were deferred to a then upcoming sprint.

6.2.1 Discussion

The four vignettes show us that the circumstances under which software developers work are in constant flux. The technology which is the both the object and tool of work is continuously updated. New people join the projects and there is learning as the newcomers begin collaborating with present members. Even the project goals themselves change as milestones are achieved. The theme that thus emerges across the four vignettes is one of ongoing and intertwined change.

Neither of the episodes were about dealing with individual or one-time changes. Whilst the vignettes began accounting for the events at certain points in time, it is meaningless to say that these were moments of change that initiated chains of events. The observer chooses suitable starting points for the accounts, but the chain of events extended from long before and certainly continued after the observations. Additionally, what is depicted are continuous changes. There was not instant and distinct change, but series of gradual changes associated with factors beyond the control of the groups under study. Furthermore, they were not discrete series of events, but converged and diverged with numerous other issues and activities. As such, they were an intricate part of daily work in both the Lambda team and the PyPy community.

In chapter 2, I outlined change as a conceptual lens for studying knowledge in practice (section 2.5.1). It emphasised the importance of how organisational workers deal with the unforeseen and unexpected (Weick 1998), and how change in organisation and practice is connected to how organisational actors respond to everyday contingencies, breakdowns, exceptions, and opportunities (Orlikowski 1996). The theme that has
emerged here affirms that this view reveals an important aspect of what software developer groups in distributed projects must cope with in their everyday work. Dealing with continuously changing circumstances is an intricate part of work practices in these settings. But the perspective on distributed software development practice offered here also highlights that this change must be analysed as ongoing. It is not instances of change that the software developer groups cope with, but evolving circumstances. As I discussed earlier (section 2.5.2.1), the computer domain is characterised by rapid and discontinuous changes in demand, competition and technology (Eisenhardt 1989; Brown and Eisenhardt 1997). I have shown how this manifests at the level of work practices in the particular context of distributed software development projects. In addition to showing that this is an important aspect of the work, I have also shown how change on this level unfolds continuously and is intertwined with the many other concerns.

6.3 Equivocal situations

The second major theme emerging through the four vignettes regard uncertainty. More specifically, uncertainty in choosing a course of action in situations that are not clearly or precisely determinable for the actors. This is about the unpredictability and indefiniteness of the situations in which the software developers have to act.

Consider how the various participants related to the depicted events. When we joined the Lambda team in the first vignette, they were debating whether to support the beta release of a new software framework version. They were uncertain because while they would eventually have to support it, there were also some issues with doing so at that point. A week later the final version of the framework update had been released and there was another debate about the issue, but the earlier decision was corroborated. However, as we saw, the situation changed radically when a remote manager reversed the decision. This meant new uncertainty, this time about what exactly had changed in the new framework version and how their product had to be adjusted. We also saw several examples of uncertainty in the aftermath of these events. Immediately afterwards, there was uncertainty about what the support of the new framework version meant for continued development, whether it was to be supported in all development streams or not. The events also increased awareness about software updates as a major
issue. Roughly two month's afterwards, there was a discussion about general strategies related to this and there was uncertainty about what would be the most appropriate.

As we saw clearly in the second vignette, the introduction of the Accord platform in the Lambda project was pervaded by uncertainty. First of all, it was unclear to what extent they were going to utilize the systems services. A lot of this uncertainty stemmed from the fact that Accord itself was also still under development (it was in a beta stage at the time of study), which meant numerous contingencies. Furthermore, as it was realized that issues were accumulating due to differing styles of usage and assumptions about functionality, there was uncertainty regarding what to do about this. We saw how there were questions about what “best practices” actually were and how to “disseminate” them within the team. Also, when these concerns were raised there was an increased level of reflection about Accord usage. Developers began asking themselves, “How do we use this?”,” Are we doing this correctly?” Another important set of issues regarded what roles were defined in the new system and how the team might have to adapt them, or adapt to them. Lastly, we also saw uncertainty regarding how to relate to the continued development of Accord itself. For instance, there were questions about if, when, and how to update to the latest version and we saw a particular incident regarding this.

Turning once again to the PyPy community and the vignette illustrating how a newcomer joined the project, it is evident that there were many uncertainties that had to be dealt with during this process. The first uncertainty facing a developer beginning to participate is where to get started. Following brief introductions, other community members are quick to offer advice, especially in terms of what online documentation seems relevant to that person. However, the newcomer is also encouraged to continue interacting with other members, asking questions and discussing their intentions in the available online forums. It is a matter, not only of the newcomer asking himself where to get started, but also of other community members asking “who are you?” so that they can relate. At the sprint, we see other examples of uncertainty needing to be worked out. Initially, it has to be decided what topics are most suitable for the tutorials and who with relevant expertise is available to lead them. Once the tutorials are finished, another discussion takes place, this time to determine where the newcomers want to begin working and, again, who with relevant expertise and shared interest is available to pair
Lastly, when the PyPy community reached the point described in the fourth vignette, there was ambivalence in terms of the project's future direction. The various contributors had different opinions and were negotiating what their joint path would be. As peripheral observers joined the discussion, there was an increased emphasis on the uncertainty as to what that path would be. The PyPy community members, knowing the potential for ambiguity or miscommunication when discussing visions in online forums (shown in section 4.3.3.4), deferred the main discussion to a then upcoming sprint. But in the online discussion, there was also an uncertainty in terms of how this was to be interpreted by observers of the community.

6.3.1 Discussion

The second theme that has emerged is uncertainty regarding how to respond to changes of various kinds. While the study participants had significant experience in their domains and a wide repertoire of knowledge, they faced situations with many contingencies and where there were many potential courses of action. Uncertainty was not a display of insecurity or indecisiveness on behalf of the software developers, but rather a matter of the situations at hand being not easily definable. This is connected to the continuous change discussed in the previous section. As these changes were not discrete, given at one point in time, but ongoing and emerging, the circumstances were not clearly or precisely determinable for the actors who had to relate to them.

There were many different actors involved, offering different possible interpretations of what had changed and options for how to relate and react to it. In short, the unfolding situations were equivocal. Thus, an important aspect of the problem illustrated in the vignettes can be described as reducing this equivocalness. With the perspective on practice assumed in this thesis (see section 2.5.1), this can be understood as ongoing processes of in situ sense-making and decisions (Giddens 1984; Weick 1995). Weick (1995) in particular has emphasised the importance of communication and interaction aimed at reducing the number of possible interpretations in organisational situations. This is what makes coordinated action possible and what we have seen are clearly instances of this. The participants needed to negotiate within the group how to relate to changes in circumstances and make decisions about a course of action. There was
uncertainty because the changes were continuous, and the following communication and interaction aimed to reduce this uncertainty to enable a coordinated response.

By highlighting how the groups under study had to act in equivocal situations, we are also touching on a topic which has implications for how we understand the underlying conditions of knowledge-sharing among sites in distributed projects. Nicholson and Sahay (2004) argued that distribution of projects brings with it the need to integrate knowledge between multiple local contexts. I argue that the findings here refine our understanding of this issue. When looking at the issue of sharing situated knowledge, analysts must also consider how ongoing change of circumstances within the project is interpreted by actors in different contexts. There are examples of related considerations in previous research. One perspective put forward emphasises standardization efforts, both in terms of technological infrastructure and of organisational practices (Sahay 2003). Others highlight the importance of social sense-making activities between the various sites (Kavanagh and Kelly 2002).

6.4 External influence

Lastly, I am going to discuss the theme of external influence that emerged through the four vignettes. At various points in the episodes, we saw how systems, events or people external to the software development groups under study influenced their activities.

The key driving force in the first vignette was the update of the framework software. Keeping up with software updates, as any programmer will know, is an integral part of software development and this entire episode was an instance of how the team coped with this issue. However, we also saw that an important point in the episode was when a remote manager intervened. This manager was the release manager of the Lambda project, but to most of the participants he was a transcendent force. He had a definite deciding role for the projects direction, but was not present on site and interacted mostly with managers and team leaders. So although an important actor in the project, he was a peripheral participant from the point of view of the Lambda development team.

The series of events depicted in the second vignette regarded the appropriation of a new platform in the Lambda project. This external component was integrated with the structure and practices of the Lambda team, requiring adjustments in both. Interestingly, we also saw how many of the participants concerns in relation to this stemmed from the
fact Accord itself was still under development. This brought with it contingencies consider. For example, a primary concern was that the Lambda team did not know how Accord was going to change in future development and thus whether there was any risk that changes would critically affect their work if they used it for source code management.

When it comes to the PyPy case, the external influence discernible in the vignettes was much more congenital. Open source communities are less clearly defined than corporate organisational projects. One dimension of this is the fleeting membership rosters. The constant coming and going of participants was first discussed in chapter 2 (section 2.5.2.2). It also emerged as a major topic in the field study findings of chapter 4 (section 4.3.3.3), and the third vignette presented in the previous chapter (number 5) showed us a particular episode where a newcomer joined the community. So external influence in this case takes the form of new participants joining the enterprise.

However, external people coming into the community is only one consequence of the transient nature of open source community. In the fourth vignette, the PyPy community were debating the future direction of the project. The key event in this debate for us was how a peripheral follower of the project joined the discussion. Indeed, several other peripheral followers also made contributions to the discussions prior, during and after this significant event, thus playing an important role in the debate. What we have is a clear example of how open source communities such as PyPy are situated in a much broader context and how this can impact directly on its activities. The PyPy project itself is a sub-set of the larger Python community (see section 4.3.1 for more on this context). The Python community, in turn, is part of the world of dynamic programming languages, meaning that there is much interrelation and exchange with other communities in the area. What we have seen in the fourth vignette is how being part of this much larger community had a direct impact on the inner workings of PyPy.

6.4.1 Discussion

The third theme that emerged in the four vignettes, then, relates to the impact of systems, events or people external to the projects under study. In general terms, a major part of the sources of continuous change stem in the fact that these projects not are isolated, but are situated in a much broader context.
In my empirical material, there are primarily two visible external influences. Firstly, there are many interrelations with other software development efforts. Software is inherently layered and composed, meaning that most software build on some pre-existing platform and uses components developed by other groups (previously or currently). Other groups may also have an interest in the project's own outcome, in terms of using the product or in terms of involvement in shared or related strands of technology. Secondly, it is also a matter of the work in a software development project being part of a much wider professional culture. There are trends in technology and practice within this culture, which influences organisation of work, decisions about how the product is to be implemented, and what tools are being used.

Such professional and technological relationships are an aspect that is not often acknowledged in research on the complexities of distributed software development. There are examples where the broader context has been considered. Nicholson and Sahay (2001) argued for a need to view the phenomenon of distributed software development “in the light of a broader social context and the backdrop of the effects of globalisation.” (ibid. pp. 41) However, the theme discussed here is on a lower level of analysis. I emphasise what being situated in a broader context means from the point of view of daily work activities. Globalisation is a prominent topic in information technology discourses, and similar calls for micro-level perspectives to understand the role of information technology in contemporary society have been made (Walsham 2001). In the context of distributed software development, however, this has been a largely unexplored perspective. Also, in addition to the primarily political and social issues framed by Nicholson and Sahay, the set of influences highlighted through my work are more specifically related to the domain of software development, in terms of both technological and professional relationships.

6.5 Chapter summary and conclusions

In this chapter, I have discussed a set of themes that emerged in the four vignettes presented in chapter 5. The vignettes illustrated episodes of evolving technology, enculturation and socialisation, and emerging project purposes and goals. Albeit they regard different subject matters – technology, participants, and goals – I have here shown how they share three themes.
The first theme discussed was that the episodes all regarded *continuous changes* that were intertwined with numerous other issues and activities over time. This affirms that dealing with changing circumstances is an intricate part of daily work in the studied settings, an aspect largely unexplored in previous research. The second theme regards that the situations in which the software developers had to act were *equivocal*, with many contingencies and possible interpretations. This refines our understanding of the underlying conditions of knowledge sharing in distributed projects, as analysts must also consider how ongoing change of circumstances are interpreted within groups in different contexts. The third theme is *external influence*, stemming in the fact that these projects are situated in a much broader context. The set of influences seen in my empirical material relate primarily to the domain of software development, in terms of technological and professional relationships. This adds to the primarily political and social issues highlighted by others, but also shifts the level of analysis towards what the broader context means from the point of view of daily work practice.

![Figure 10: Three aspects of the changing circumstances.](image)

My conclusion is that these three themes characterise how change of circumstances unfold in practice in the two settings that I have studied. They are closely interrelated, representing three integral aspects of the situations facing the software developers, as illustrated in Figure 10 above. Having discussed how changing circumstances unfolds, what emerges is an image of distributed software development as an environment in constant flux. There is continuous change, associated with external influence and resulting in equivocal situations, which the groups constantly must relate to in their work. Software developers groups have to act within this complex web of contingencies extending well beyond the boundaries of the specific projects. In the next chapter, I will
turn attention towards what my empirical analyses have unearthed regarding how the software developer groups under study dealt with these issues.
7 Framing the purposeful adaptation of practice as knowledge development

7.1 Introduction
This thesis focuses on the issue of how software developer groups in distributed projects continually develop their practical knowledge. In order to do so, I have presented empirical material on work practice based on two extensive field studies and explored in particular how the software developers dealt with changing circumstances. In this chapter, I will shift attention back to the issue of knowledge, discussing how my findings on the work practices in the Lambda team and the PyPy community addresses the thesis objective.

First, I will put forward evidence from my empirical material for what I call the purposeful adaptation of practice in response to changing circumstances. Based on my discussion of what characterises how changing circumstances unfold in chapter 6, I argue that this is an ongoing process with at least two concerns: anticipation and alignment. The concrete exemplification of this process through my research work is discussed in relation to the perspective on practice emphasising emergence and improvisation I set in chapter 2, and I also discuss the role of agile methods.

Next, I argue that, from a perspective where knowledge and practice is seen as evolving in conjunction, the purposeful adaptation of practice can be seen as one of the processes by which software developers in distributed projects continually develop their practical knowledge. As they continually adapt their practice in response to changing circumstances, they are also updating what they know about how the work can be done within particular contexts. I conclude by discussing how my findings contribute to
literature on situated knowledge in distributed software development.

### 7.2 The purposeful adaptation of practice

#### 7.2.1 An ongoing process

The argument that I make here, is that the software developer groups under study dealt with changing circumstances through what I call the purposeful adaptation of practice. I mean that practice – that is, the recurring coordinated activities of groups doing work as informed by a particular organisational or group context (the definition used in this thesis, presented in section 1.4) – becomes suitable for present opportunities and constraints by being constantly adapted to changing circumstances. This is a purposeful process, I argue, because it is the result of reflection and deliberation among the software developers. That is to say, it is not accidental evolution - the consequence of natural selection on random mutation - as there is intent and conscious choice behind it.

Evidence from my studies of work practice and changes in technology, participants, and goals in the Lambda team and the PyPy community make visible two features of purposeful adaptation.

First, practices that have been adopted specifically to allow for changing circumstances. Simply put, the groups expect that some circumstances – I have specifically looked at technology, participants, and goals - are going to keep on changing. Therefore, they adopt general approaches such as agile methods and practices for coping with the particular changeable factors of their specific situation. The following examples from the preceding empirical analyses illustrate this:

1) In both settings, agile development methods were applied.\(^{36}\) They allowed the groups to work towards emerging project goals, which were reliant on, for instance, changes in available resources (Lambda team), or what priorities the community's core group agreed on (PyPy community). This, of course, relates to the application of approaches well-known and exercised in the domain – neither Scrum, eXtreme Programming, nor sprint-driven development is unique to the cases at hand. But I have shown how they were applied to fulfil this purpose within particular contexts, which included fitting them to specific situations. For

\(^{36}\) Sections 4.2.1 and 4.3.1.
example, PyPy's sprint-driven development is their own variant, and the Lambda team were actively working out how to best achieve the goals of Scrum meetings.

2) One of the important purposes of PyPy's sprints was as a strategy for attracting newcomers and a way of facilitating a process of enculturation that enabled them to become competent participants. But, importantly, we have also seen, particularly through the third vignette in chapter 5, how PyPy's practices relating to newcomers specifically allowed for a tailored introduction. Not only was the practice intended to deal with the transient nature of the community, it also acknowledged each newcomers individual backgrounds, interests, and potential as contributors.

3) I have highlighted how sprints were a way to work out intricate technical details and make overarching design decisions. Pertaining to this, we also saw an example of a complex, controversial discussion being deferred from online forums to an upcoming sprint in connection with external influence. This was an important element in PyPy's practice for dealing with a constantly emerging project direction.

4) The Lambda team making exploratory events, training courses, and technical seminars an intricate part of their daily work is another example. Their intention was to adopt work practices that enable the participants to continually develop their software development skills and hence make sure that they could deal with the ever evolving field of software technology.

Second, changing circumstances stimulating adjustment of practice. When unforeseen changes occur, there is reflection within the groups – observable in the discussions that take place in connection with events - on what to do (or not to do) about it. Sometimes practice is amended or changed following this (although sometimes the choice is to do nothing). Evidence for this can be found throughout my empirical analyses:

37 Particularly section 4.3.1.2.
38 Section 4.2.3.3.
39 Section 4.3.3.3.
40 Section 5.5.
41 Section 4.3.3.4.
42 Section 5.6.
43 Section 4.2.3.4.
5) In the fourth vignette presented in chapter 5, the PyPy community were having a discussion about what areas to explore in the future following a major release. These were complex and diverging discussions, and as there was external pressure in the form of peripheral followers of the project making opinionated entries, the PyPy core groups implicitly decided to defer the discussions to an upcoming sprint, as they were used to doing. However, it also came to light that the lack of response could be interpreted as lack of direction by the peripheral followers, so PyPy core members needed to amend their practices in this particular instance by explicitly communicating their intentions.

6) The Lambda team underwent some significant organisational changes at the beginning of my study: the project goals changed, available resources were reallocated, and the team was partially re-composed. Subsequently, they were spending time actively working out how to best achieve the goals of team activities within this particular group in their particular situation. They tried various formats and structures, and subsequently discussed what was satisfactory.

7) In chapter 5, there were two vignettes about episodes of changing technology in the Lambda team. In both of these, important segments regarded the teams discussing their situations and how to relate to the various contingencies that came to light, particularly in terms of future work practice strategies for dealing with issues of similar nature.

8) With the introduction of Accord in the Lambda project, the Lambda team came to reflect on the tasks and roles within the team in relation to the constraints that the new platform introduced. Indeed, I also showed how the team leaders identified a need for people to alter their routine in terms of how they were using the collaboration platform.

---

44 Section 5.6.
45 Section 4.3.3.4.
46 Section 4.2.1.1.
47 Section 4.2.3.3.
48 Section 5.3 and 5.4.
49 Section 5.4.
50 Particularly section 5.4.4.
51 Particularly section 5.4.2.
9) It is important to mention the PyPy community's highly self-aware and reflective application of sprint-driven development, which has been shown throughout the empirical material presented in this thesis. The PyPy core group continuously discussed their methods and made adjustments to it, something that is exemplified particularly through the extracts in section 4.3.3. For instance, there were examples with comments on the project rhythm, collocated discussions on design questions, and organisation rules for sprints. 

Adopting practices to allow for changing circumstances and changing circumstances stimulating adjustment of practice are tightly interlinked. They are two aspects of how the software developers make their practice suitable for present opportunities and constraints, a combined process. The adoption of practice that allow for changing circumstances can itself come about through reflection and adjustment. For example, we have seen how the PyPy community over time had come to realize the potential and make use of interwoven online and collocated events, and how the incidents of evolving technology caused the Lambda team to begin developing strategies for dealing with such issues in the future. Furthermore, whereas general agile methods obviously are adopted, we have also seen how they can be gradually tailored and adjusted to fit particular situations in both the PyPy community and the Lambda team.

7.2.2 Two important concerns

The purposeful adaptation of practice happens while the software developer groups engage in their endeavours, an ongoing process in response to the continuously evolving environment in which they work. Discussing how changing circumstances unfold in chapter 6, I highlighted three integral aspects of the situations facing the software developers: continuous change, equivocal situations, and external influence. Based on these insights, I now argue that the purposeful adaptation of practice in dealing with changing circumstances – the ongoing process highlighted above – involves at least two important concerns: anticipating changing circumstances and aligning activities within the groups.

7.2.2.1 Anticipation

The first concern is anticipation. I mean anticipation in an active sense, so as to nullify, prevent, or forestall the impact of something by realizing and considering it beforehand.
As seen in my empirical material, this anticipation is either the subject of reflection – visible in discussions - following unforeseen change of circumstances, or embedded assumptions in the practices adopted to allow for changing circumstances. In the following, I will exemplify instances of such anticipation that have been unearthed in my work by placing them in relation to the three themes highlighted in my discussion of how change in circumstances unfolds (see chapter 6).

Through my studies of Lambda and PyPy, it has been seen how the software developers have to anticipate what the continuously changing factors are:

1) The practices adopted by the PyPy community for receiving newcomers anticipated that everybody approaching the community would have different backgrounds and interests, in that they aimed to allow these uncertainties to be addressed.\(^{52}\)

Furthermore, the software developers have to anticipate what implications a chosen course of action will have, since the situations they find themselves in are equivocal:

2) When considering how to deal with the actual installation and maintenance of the Accord platform, the Lambda team had to anticipate how people could handle it taking into account their current responsibilities.\(^{53}\)

3) Interestingly, in the vignette illustrating how the PyPy community came to defer an online discussion to a then upcoming sprint, the practice of doing so in that particular situation actually failed to anticipate that it might be interpreted as “answering by silence” in the online forums.\(^{54}\)

The software developers also have to anticipate the impact of external influence, among them what the technological and professional relationships are and how to relate to them:

4) When deciding on what areas to prioritize in future development, the PyPy community were particularly anticipating external expectations, discussing explicitly their own experiences of interacting with users and how the product was actually used.\(^{55}\)

\(^{52}\) Section 4.3.3.3 and 5.5.  
\(^{53}\) Particularly section 5.4.2.  
\(^{54}\) Section 5.6.  
\(^{55}\) Section 5.6.
5) In both the software framework and the Accord vignettes, the Lambda team members were anticipating contingencies when deciding how future updates should be handled, based on their experience of having a prior decision overturned.\textsuperscript{56}

### 7.2.2.2 Alignment

I have shown how changing circumstances involves continuous change and how, at the same time, the participants in software developer groups do not cease their work and rethink practice every time there is a change. Rather, they continue with their activities while the conditions are evolving. As a group, they may be given directions (more likely in an organisational setting) or plan \textit{in situ} for how activities need to be altered. But, as has also been shown, they will constantly be facing equivocal situations, open to interpretation. I argue that the second important concern of the purposeful adaptation of practice is gradual alignment of individuals activities following change, either spontaneously following unforeseen change or as part of adopted approaches. The following examples – again put in relation to the three themes highlighted in the previous chapter – illustrate this.

The first examples regard how activities were aligned in the face of \textit{continuous change}:

1) How the Lambda team were learning to use the Accord platform consistently within the group is a primary example of this alignment, as this occurred when practical problems accumulated due to slightly differing practices when using Accord and the participants negotiated a common road as they realised these conflicts.\textsuperscript{57}

2) Alignment was facilitated by the regular sprints in PyPy, as they provided a rhythm for the work in the project, and, in combination with sprint practices aimed at maximising time spent working \textit{together}, this ensured synchronous work during the actual collocated events and that the participants could work in an aligned fashion between them.\textsuperscript{58}

\textsuperscript{56} Sections 5.3 and 5.4.
\textsuperscript{57} Section 5.4.
\textsuperscript{58} Section 4.3.3.1.
Furthermore, there are examples of the software developers aligning in a stance regarding *external influence*:

4) In the vignette illustrating how the PyPy community deferred a discussion to an upcoming sprint, a project facilitator stepped in to communicate their intentions when the peripheral followers asked about what was happening. Following this, there were several responses from other core members who added to this line of reasoning, thus aligning with the project facilitators initiative.\(^{59}\)

5) Towards the end of the vignette illustrating the introduction of the Accord platform in the Lambda project, there was an incident where an update of the servers failed and caused a staggering of the workflow in the Lambda team. One of the consequences of this breakdown was that the assumptions about when it was appropriate to update had to be aligned among the group members.\(^{60}\)

Lastly, we find examples of aligning with the groups chosen course of action in an *equivocal situation*:

7) Following the main events of the incident with the framework update, there was a period of alignment of efforts as awareness of the final solution took some time to spread outside the group of developers directly involved.\(^{61}\)

8) When a newcomer attends his or her first PyPy sprint, time is spent at the very first status meeting to decide if there is a wish for any introductory tutorials and, if so, what the topics are to be and who with relevant expertise is available to give it. The discussion about topics are a way of aligning expectations among newcomers and core members as to what is relevant and what is to be achieved in the forthcoming development activities.\(^{62}\)

Of course, this is not to say that there necessarily _has_ to be alignment. We can take an example from the open source world: if there is strong disagreement within a project, and the circumstances allow, then a branch in the development might be created. But I am saying that if there is a continued coherent group effort, then within that particular group there will necessarily be gradual alignment.

\(^{59}\) Section 5.6.3.  
\(^{60}\) Section 5.4.5.  
\(^{61}\) Section 5.3.2.  
\(^{62}\) Section 5.5.2.
7.2.3 Discussion

To summarize, I am suggesting that one of the ways which the software developers in my studies dealt with changing circumstances was through what I call the purposeful adaptation of practice. I have reiterated evidence for such a process unearthed throughout my empirical material, showing it as a combination of practices adopted to allow for changing circumstances, and changing circumstances stimulating adjustment of practice. It is purposeful - the result of reflection among the software developers, visible in discussions - and I have shown how there are two important concerns in how it is achieved: anticipation and alignment. These two concerns are the driving forces through which the purposeful adaptation of practice continually happens in response to changing circumstances as the software developers are engaging in their work. Figure 11 below shows the relationships of this process.

![Figure 11: The purposeful adaptation of practice.](image)

In this thesis, I have used a lens on practice that emphasises emergence and change (introduced in section 2.5.1). My intention with this was as a way of talking about the continual development of knowledge by focusing on what software developers do, their actual practice (I will return to this relation in the following section). I put forward the concept of improvisation, which has been suggested as a way of understanding how workers deal with the unforeseen and unexpected (Weick 1998; Ciborra 1999). I consider my idea of the purposeful adaptation of practice as a concrete description of
Chapter 7 • Framing the purposeful adaptation of practice as knowledge development

one particular process of improvisation. In particular, my explanation of how software developers deal with changing circumstances through the purposeful adaptation of practice retains a balance between the routine and the spontaneous, something that is at the heart of improvisation. It acknowledges how “order can be accomplished by means of ongoing ambivalent mixtures of variation and retention that permit adaptation to dynamic situations” (Weick 1998 pp. 551). I have also concretely shown how what Orlikowski (1996) calls “improvising organisational transformation over time” happened as the software developers responded to everyday contingencies, breakdowns, exceptions, and opportunities. My contribution to the idea of improvisation in the particular settings under study – Lambda project and PyPy community - is to show how anticipation and alignment are two of the key concerns in how the software developers deal with continuous change, equivocal situations, and external influence.63

A comment must also be made about the role of agile methods in what I have seen. Agile methods – including Scrumm, eXtreme Programming, and sprint-driven development - were utilized in both the Lambda and the PyPy case. But I have also shown how they were specifically modified to particular circumstances and suggested that the agile methods themselves are constantly adjusted as there is change in those circumstances. While my research has not focused on issues of method, this does echo what empirical research on tailoring of agile methods have found (Fitzgerald, Hartnett et al. 2006). Fitzgerald, Hartnett, et al. write that “at a micro-level, the XP and Scrum methods provide an overall framework within which micro-level tailoring occurs at individual project level” (ibid. pp. 209). In their case study, it emerged that the software developer groups selected parts of methods and also modified some practices and replaced others with substitute practices. What my findings add is also putting the spotlight on the connection between ongoing change in circumstances and this tailoring of methods.

63 It should be noted that the terminology of my work on the purposeful adaptation of practice bears a resemblance to Hutchins ideas about the social organisation of work through adaptation (Hutchins 1991). However - as was pointed out in section 2.5.1 - while I draw on practice-based traditions, Hutchins work is within the tradition of distributed cognition (Hutchins 1995a; Hutchins 1995b). Whereas he is interested in the adaptive capabilities of organised groups seen as computational systems - particularly in terms of cognitive questions - I am focusing on how software developers can be observed to adapt their practice. There is therefore a difference in the level of analysis. Another difference lies in our analytical focus. I am specifically talking about how the software developers reflect on their situations and take action, while Hutchins talk about adaptation on the system level without explicit intention in action. Hutchins work is a useful reference to be aware of, but it is outside the scope of this thesis to delve further into the similarities and differences of the two perspectives.
7.3 Purposeful adaptation from a practice-based perspective on knowledge

7.3.1 Rephrasing adaptation of practice as knowledge development

In this thesis, I have assumed a practice-based perspective on knowledge (introduced in section 2.3) in order to address the issue of how software developers in distributed projects continually develop their practical knowledge. This perspective centres on knowledge as a person's ability to engage in a practice, something that is intricately tied to the context. This implies that practical knowledge is enacted and developed in response to changes in that context. I therefore argued (in section 2.4) that one fruitful avenue for investigating how practical knowledge is developed is to focus on how the software developers deal with changing circumstances. Based on literature on the complexities of software development as an activity in connection with my perspective on knowledge and practice, I suggested that evolving technology, enculturation and socialisation, and emerging project purposes and goals are relevant topics to explore for such an investigation. Building on the empirical findings that emerged through my studies, I have now suggested the purposeful adaptation of practice as an explanation for how the software developers deal with changing circumstances.

What remains now, is for me to tie back to the overarching issue of knowledge development that I am addressing in this thesis. From the viewpoint on knowledge assumed here, human action has epistemic value in itself, as it is needed to account for what someone knows (Cook and Brown 1999). This thesis thus speaks of what the software developers under study know about how the work can be done, their ability to participate in practice within these particular contexts of distributed projects. What they know is intricately tied to what they do, and we have seen how they evolve what they do through the purposeful adaptation of practice. Ergo, practical knowledge and practice evolve in conjunction. Gherardi expressed this as that “participating in a practice is [...] a way to acquire knowledge-in-action, but also to change or perpetuate such knowledge [...]” (Gherardi 2001 pp. 215). This is why I argue that what has been shown through my field studies and subsequently discussed in this chapter can be interpreted as practical knowledge being enacted and developed as the software developers engage in their daily work.
The purposeful adaptation of practice is one way that the software developers continuously make sense of their situation and put what they know into practice. As they deal with continuous change, equivocal situations and external influence, they rethink and extend what they know. One way to understand this as a knowledge-related issue is to speak of it in terms of building up, maintaining, and improving competence. First, it is a way for the group to cope with their task despite ever-changing conditions and constantly maintain and improve their efficiency, or productivity, or creativity, or whatever is valued in their particular context. This is obvious when looking back at the examples of section 7.2.1 above. For instance, the Lambda team members were rethinking what they knew about how to best achieve the goals of team activities following changes in team composition and resources (example 6), so as to be able to fulfil the expectations within their organisation. The PyPy community's constant reflection on and improvement of the application of sprint-driven development (example 9) is another illustrating instance, as they explicitly discussed what they knew and what they learned through their experience of sprinting. Second, it is a way that the individual developers constantly build up their abilities as practitioners, gaining experience about a variety of practice-based issues that they can bring with them to future projects. We can see the importance of this in how large a role previous experience played in the purposeful adaptation of practice. This is particularly clear in examples 2) and 7). Lambda team members related to new technology according to previous experiences, but also had to adapt what they knew to fit within the particular team and their current situation. Similarly, newcomers brought with them ideographic expertise and competence to the PyPy project, and the practice of introducing them looked especially to how that knowing could be integrated within the particular project at hand.

Another aspect that is important to take into account when discussing knowledge-related issues through purposeful adaptation, is how the process that I have highlighted evidence for in this chapter is constantly ongoing, embedded in practice and the result of reflection among the software developers in relation to changing circumstances. In particular, I have argued that it concerns constant anticipation and alignment among the participants. What this signifies is constant learning in practice, articulating the dynamics through which this happened in the two settings I studied, namely the Lambda
team and the PyPy community. The ideas of learning in practice – which I discussed in section 2.3.3 – stress that it is a social process, something continuously achieved by acting and interacting within communities of practice (Lave and Wenger 1991; Wenger 1998). The interacting and acting involved in actively changing what you do in response to constantly changing circumstances – what I have described in my work - manifests in anticipation and alignment. The examples of sections 7.2.2.1 and 7.2.2.2 thus show us concretely the ways that the software developers go about maintaining their competence and tells us how they develop what they know in order to do so. For instance, in examples 1) and 2) of section 7.2.2.1 we saw the importance of knowing the technological areas of the project as well as the plans and activities in relation to these. This was the case when addressing what newcomers could engage in appropriately and how people could handle new technological contingencies with their current responsibilities. Examples 4) and 5) of section 7.2.2.2, on the other hand, illustrate how there was change in what the software developers knew about appropriate practice in connection with breakdowns. PyPy community members realized that they had to communicate their intentions more clearly in their practice of deferring complex discussions, while Lambda team members had to reconsider their practice of updating the infrastructure.

We can re-phrase the argumentation of this chapter, then, in terms of practical knowledge being developed. Software developers working in groups in distributed projects know ways of dealing with circumstances that are continuously changing, and learning takes place in response to breakdowns as circumstances change and practice must be adjusted. Two important elements of this process is alignment of practice and anticipating change, which articulates a process of learning in practice through negotiating shared approaches for doing things and being cognisant of the implications within the team of actions and decisions relating to change.

7.3.2 Discussion

The purpose of my thesis has been to complement previous research on knowledge-related issues in distributed software development. Whilst issues related to the situated aspects of knowledge have been put forward in literature (Sole and Edmondson 2002; Nicholson and Sahay 2004; Imsland and Sahay 2005), I argued – in section 2.4 - that
there was a further need for in-depth, empirically-grounded investigations of how knowledge is enacted and developed in particular contexts within distributed projects. By assuming a practice-based perspective on knowledge, I have addressed this need with my studies of work practice in the Lambda team and the PyPy community.

My contribution to the literature is that I unearth more of the dynamics which underlie knowledge development in these settings. With the purposeful adaptation of practice and its two concerns of anticipation and alignment, some of the underlying conditions for how knowledge sharing actually happens in distributed software development projects are unveiled. Nicholson and Sahay (2004 pp. 358) pointed out that through a “process of negotiation can locally specific knowledge around organising principles be understood by the various actors involved, and revised approaches developed.” What I have done is to show concretely how practical knowledge is developed as the software developers purposefully adapt their practices, shifting the focus of analysis towards how such revision of approaches actually happens in practice, as compared to the negotiation between organisational partners in outsourcing relationships discussed by Nicholson and Sahay (ibid.). One implication of my perspective is that the temporal dimension of knowledge is stressed further, as it shows how practical knowledge is not stable, but constantly developing in connection with changing circumstances. It also implies that aspects related to intent, awareness, and reflection are important in how software developers in distributed projects enact knowledge in practice. Whereas previous research has put forward mechanisms and consequences, this thesis provides an image of “reflective software developer groups”, shifting our attention towards supporting their contemplative activities.

Furthermore, in relation to my findings about distribution as a mundane issue (see chapter 4 in particular), as something that the software developers simply factor in with other issues, my contribution on how practical knowledge continually is developed in practice opens up for a change in focus: from what distance means, to what situated means in distributed software development. Sole and Edmondson (2002 pp. 18) wrote that “situated knowledge – knowledge grounded in site-specific work practices – plays a critical role in dispersed team learning.” I suggest that an alternative view is to also consider that knowledge may be situated in shared practices and the process of purposeful adaptation, which may or may not be tied to particular locales.
In the Lambda team, there were some degree of locale-specific practices as the team was working relatively independently within the project, but there was also fluent exchange with other teams through meetings and available information technology infrastructure. This meant that interaction with distant colleagues simply was perceived as an inherent part of the work. We can describe situated knowledge in this case as a matter of a shared professional context, albeit the purposeful adaptation of practice that I observed was constrained to the particular site of the Lambda team (in correspondence with previous research, as also flagged in section 5.4.6). In the PyPy community, the interweaving of online and collocated events meant that a shared practice and - to a much larger extent that in the Lambda team - a shared process of purposeful adaptation was present and maintained within the whole community (who were, of course, dispersed). That knowledge was situated there meant within a particular shared enterprise rather than in specific locales.

One important implication of this, is that if we consider situated in terms of professional contexts or shared enterprises, then it is imperative that there is more explicit discussion about practice and changes through available forums. This was visible in both my cases. In PyPy, the community spent much time on articulation work both online and at sprints, explicitly discussing how they were doing the work and, importantly, why. A pertaining example from the Lambda study is how a member of the development team with a supporting role was approached differently by local and remote colleagues (discussed in section 4.2.3.2). Whereas the local colleagues would assume that - or at least know when - she was available, the remote colleagues required her to explain in detail the reasons for why certain solutions worked.

**7.4 Chapter summary and conclusions**

In this chapter, I have addressed the issue of how practical knowledge is continually developed among software developers in the two distributed projects that I have studied, fulfilling the objective of my thesis. This was seen in how the software developer groups under study dealt with changing circumstances in practice.

My argument is that this happened through what I call the purposeful adaptation of practice, a combined process of practices adopted to allow for changing circumstances and changing circumstances stimulating adjustment of practice. Extracting evidence
from my empirical analyses (chapters 4 and 5), I have shown concrete examples of this process. Combining the characterization of how change in circumstances unfold from chapter 6 with further evidence from my empirical material, I also showed how it involves at least two concerns: anticipation and alignment. These two concerns are the driving forces through which the purposeful adaptation of practice happens in response to continually changing circumstances. (The relationships of this ongoing process are shown in Figure 11 on page 145.) Discussing these arguments, I suggested that they are a way of describing improvisation within the particular contexts of distributed software development, linking my findings back to the perspective on practice introduced in section 2.5.1. In particular, it is about retaining a balance between the stable and the emergent aspects of work. Also, I discussed the role of agile methods in the purposeful adaptation of practice. Conferring with empirical studies on how agile methods are often tailored by practitioners, the suggestion was that they provide a framework which are modified and complemented according to the particular project situations.

Shifting attention back towards the issue of practical knowledge, I then reintroduced a practice-based perspective on knowledge which entails that knowledge and practice evolves in conjunction. From this position, I argued that the purposeful adaptation of practice can be rephrased as one of the processes through which the software developers continually develop their practical knowledge. They know ways of allowing for changing circumstances, and learning takes places as breakdowns occur and adjustment is necessary. In discussing this argument, I proposed that this articulates the dynamics of how constant learning in practice happens in the context of distributed software development projects. Finally, I discussed my contribution to the literature on knowledge-related issues in distributed software development. By unveiling the dynamics of how practical knowledge continually is developed in response to changing circumstances, the temporal aspects of knowledge in distributed software development settings are put forward and there is a shift in focus towards supporting software developers thoughtful practices as aspects related to intent, awareness and reflection are emphasised. Lastly, my empirical findings opens up for an alternative understanding of what in means that knowledge is situated in distributed software development projects, as both the Lambda and the PyPy cases illustrate how this can be as much a matter of professional context and a shared enterprise, as of specific locales.
8 Conclusions and reflections

8.1 Summary of thesis structure

Chapter 2 provided a literature review on issues of knowledge and change. Looking at literature on knowledge-related issues in distributed software development, I identified a need for further research into how practical knowledge is enacted within particular contexts. Outlining a practice-based perspective on knowledge, I contended that one way of investigating this was in how software developer groups deal with changing circumstances. Extending the review to literature on the complexities of software development as an activity, I suggested that the topics of evolving technology, enculturation and socialisation, and emerging project purposes and goals were relevant venues to explore for this purpose.

Utilizing material from two extensive field studies, I analysed work practice in two different settings of distributed software development: one development team in a multinational organisation and one open source community. The methods of the field studies were primarily participant observation coupled with grounded theory coding techniques. Chapter 3 outlined the research context and methodological framework, while chapter 4 presented the specific research processes and elucidated practice through the analytical categories that emerged. This provided a first set of evidence for my thesis argument.

In chapter 5, I extracted four episodes from the empirical material where the groups encountered changing circumstances in terms of technology, participants, and goals. They were rendered into vignettes, illustrating the three topics identified in chapter 2 within the particular contexts under study. This provided a refined view on how these issues were dealt with in practice, providing additional evidence for the thesis argument.
and an empirical context for the discussion of changing circumstances.

To build a basis for understanding how the software developer groups deal with changing circumstances, I proceeded with a discussion of the shared themes that emerged in the three vignettes in chapter 6. This helped me describe how changing circumstances unfold in general terms and hence what situations that faced the software developers.

Finally, in chapter 7, I brought the analysis back towards the issue of how software developers in distributed projects continually develop their practical knowledge. Utilizing the empirical material from the preceding chapters, I highlighted evidence for what I called the purposeful adaptation of practice in dealing with changing circumstances. To conclude the inquiry, I rephrased this as one of the processes through which software developers in distributed projects continually update what they know about how the work can be done in particular contexts. The implications and contributions of this were discussed, particularly in relation to literature on situated knowledge in distributed software development.

### 8.2 Thesis conclusions

In this thesis, I have investigated how software developers in distributed software development projects continually develop their practical knowledge. The term “practical knowledge” is used in reference to what the software developers know about how the work can be done within particular contexts. I contend that one of the ways that this knowledge is continually developed is in dealing with changing circumstances, referring to the constraints, opportunities, and resources which the actors can dispose in their activities. Thus, the research question addressed is:

- **How do software developers in distributed projects continually develop their practical knowledge through dealing with changing circumstances in daily work practice?**

I have found that the software developers are facing continuous – as compared to discrete or distinct – change, associated with external influence and resulting in equivocal situations. I argue that one of the ways that they deal with this is through what I call the purposeful adaptation of practice. This is a ongoing process of practices being
adopted to allow for changing circumstances and changing circumstances stimulating adjustment of practice. It is a reflective process, driven by at least two important concerns: 1) anticipating changing circumstances, and 2) alignment of activities. I conclude that the purposeful adaptation of practice is one of the processes through which software developers in distributed projects continually develop their practical knowledge, and hence that the research question has been addressed.

8.3 Contributions

8.3.1 A view on distribution as a mundane issue
The objectives of the thesis were to focus on work practice in the context of distributed software development. Two field studies provided the empirical source material. Throughout the thesis, I elucidated the practices unearthed in those studies. In the first case, the workaday of distributed software development in a multinational organisation was illustrated, while in the second it was the collaborative practices of an open source community that were studied.

My contribution with this empirical material has been to show distribution as a mundane issue. In the open source study, it was shown how sprint-driven development facilitated synchronous work and project rhythm, strong social relationships among members, attraction and enculturation of new members, and an ability to defer complex or controversial discussions. That is, the interweaving of collocated events and online activities served more specific purposes within the community than simply alleviating distance. In the organisational setting, issues of collaborative problem solving, adjustment of practice for team activities, informal and non-work task related activities, and relating to uncertainties and change all appeared alongside interaction and coordination with remote colleagues. The distributed dimension of the project played a minor role in everyday work practice and was not viewed as a distinct problem.

This helps shed further light on the complexities of distributed software development. The image that emerges contributes a different perspective on the issue of distribution, one where distribution is simply one aspect of the setting - something that is factored in with other, more specific, issues. It is a different way of talking about the phenomenon, one that replaces generalities and distinctions in favour of the gritty details of everyday
activities. What I have revealed is how software developers cope with distribution among other issues in particular settings, what the major constraints are, and what resources they utilize in overcoming them.

8.3.2 Characterising changing circumstances
A central proposition of this thesis is that practical knowledge is developed as software developer groups deal with changing circumstances. Three topics were suggested as relevant for exploring how this happens in distributed software development: evolving technology, enculturation and socialisation, and emerging project purposes and goals. My contribution has been to provide detailed empirical material that refines our understanding of how these issues manifest themselves in particular contexts of distributed software development.

Furthermore, through my discussion of the shared themes between the different vignettes, I also characterised how changing circumstances unfold in distributed software development settings. In doing so, I have shown that dealing with change is an intricate part of daily work practice in such settings and what the major constituents of this problem are. This has highlighted an aspect of distributed software development practice largely unexplored in previous research. It also puts forward additional elements of the context in which distributed software development takes place, adding technological and professional relationships to the primarily political and social issues previously discussed.

8.3.3 Extending research on situated knowledge
Lastly, the specific aim of this thesis is to make a contribution to the discourse on knowledge-related issues in distributed software development. The contribution has unveiled a novel aspect: how practical knowledge is developed as the software developers deal with changing circumstances. Rather than viewing knowledge in a static sense, this highlights the temporal dimension of knowledge in these particular settings.

The image that emerges through this work is that of “reflective software developer groups”. It contrasts with the previous imagery of knowledge-related issues in distributed software development, which mainly accentuates the mechanisms of knowledge sharing within the project and the consequences for the enterprise if it is
lacking. Here, instead, the key feature is how the software developers reflect on their situations and relate to ongoing changes in circumstances. They are aware of their own and others role within the project, cognisant of the implications of actions and decisions. This refines our understanding of the conditions of knowledge sharing in distributed software development projects, as it uncovers the underlying contingencies that matter for how learning happens through the purposeful adaptation of practice.

Lastly, my findings open up for an alternative understanding of what situated means in distributed software development. I suggest that, in addition to locale-specific practices, it can also be understood in terms of professional context and a shared enterprise. The ongoing process of purposeful adaptation of practice can be shared among the members of a distributed group, although it requires more articulation work within available forums.

### 8.4 Areas for further research

The purpose of this research has been to make an empirical contribution to the subject of distributed software development practice. There are now a number of potential areas for extension and development of this work.

First of all, whereas I have discussed my contribution to the literature on situated knowledge in distributed software development, I recognise that there is a need to integrate my thesis with other common topics in the area as well. I envisage new avenues of research being opened, particularly concerning the temporal dimension of knowledge and alternative understanding of what situated means. The impact on our understanding of the maintenance of mutual knowledge (e.g. Cramton 2001) and awareness of current project status (e.g. Damian, Chisan et al. 2003) throughout the project is one area for further research. Another is how this adds to how we understand differences in national (e.g. Krishna, Sahay et al. 2004), organisational (e.g. Damian, Izquierdo et al. 2007), managerial (e.g. Sahay 2003), and practical (e.g. Tellioglu and Wagner 1999) cultures between the various sites involved in the project.

In terms of the results from my own work, I believe there are a number of themes that could be extended. To increase the practical applicability of my findings, there is a need to look into the role of information and communication infrastructures in mediating the purposeful adaptation of practice, as this is an aspect that I have not yet given specific
attention. Furthermore, the role of agile methods needs to be elaborated upon. I briefly tied in with empirical research on the tailoring of agile methods (Fitzgerald, Hartnett et al. 2006) in my discussion of the purposeful adaptation of practice, but there is a need to further consolidate my findings in the context of this research area (Dybå and Dingsøyr 2008). One potential topic that could be explored with my empirical contribution is what the principles of agile methods are actually supporting in practice - something that has been flagged as a topic of future research within this community (e.g. Wang, Lane et al. 2009). Also, the topic of agile methods in distributed software development (e.g. Paasivaara and Lassenius 2006) and open source communities (e.g. During 2006a) is gathering attention and I believe this to be another fruitful area in which to extend my research.

Lastly, there is a need to further explore the purposeful adaptation of practice in distributed software development. In this thesis I have only begun to highlight this process and its key aspects, but the image of how it happens must be made more detailed if we are to devise ways of supporting it, particularly in terms of interaction and communication among the software developers for the purpose of adapting practice. Potential questions are: *What resources the software developers draw on when anticipating changing circumstances?* and *How are differences in practice in relation to changed conditions communicated when alignment occurs?* There is room for further analysis of such questions using my empirical material, as the qualitative approach and participant observation method has made it rich and extensive. However, because of the breadth and exploratory nature of my field studies, I also recognise that there is a need for further empirical research that is directed specifically towards these issues.

### 8.5 Limitations and lessons learned

One source of potential critique against my findings and conclusions is the selection of, and relationship between, the two cases. First of all, I have described how the selection of cases was opportunistic, which might raise concerns about their representability. But this must be seen in the context of pragmatic academic work and qualitative methodologies. Gaining access to real-world settings is difficult, and researchers are – as in my case – often reliant on personal connections and what opportunities present themselves. But this does not affect the validity of the studies. The strength of an
interpretive approach is the ability to explain ideographic cases, and the important factor in the selection of cases is a fit with the research focus. Both my cases represent settings that I set out to study, namely distributed software development in real-world contexts. Nor does it affect generalizability, because the generalisability that we are concerned with in qualitative research is not from individual cases to a larger population (a claim that itself rests on volatile philosophical ground due to the problem of induction), but from empirical descriptions to theory (Lee and Baskerville 2003).

I have already discussed the differences between the two settings under study (see especially sections 1.2, 3.4.4 and 4.4). But one must also consider the differences in study process. The Lambda study was comparatively concentrated, taking place across 5 months and focusing on collaborative work practice in this one particular site. Participant observation was the method of data generation. In contrast, the PyPy study was much more extensive and exploratory. It went on for over 2 years, during which time there were multiple shifts in focus and a wide range of methods were utilized. However, the differences in process between the two studies are not an inherent weakness for my results, because the purpose has not been to compare the two cases. Despite obvious differences in settings and study processes, evidence for the purposeful adaptation of practice is present in the empirical material from both studies.

An important element of both my studies was the maintenance of an active relationship with the study participants. This both enabled my research in terms of access and directly contributed to the production of results. Of course, one may also raise the potential issue of bias with this; of whether there was purposeful selection of cases and if my findings simply confirm the beliefs held by those under study. Addressing such concerns, one must first point out that with interpretive research the subject is people and the social reality which they enact. Ethnographic methods are intended specifically to access the perspective of those under study and interaction with them is essential to this. Having said that, the researcher must also be sensitive to systematic distortions in the narratives collected from the participants, what Klein and Myers (1999) referred to as the principle of suspicion. I have taken practitioners input as cues to what issues were important and have used it as evidence. But I also drew on numerous other sources in my analysis, including my own empirical observations, collected artefacts, and views expressed by a wide variety of commentators. Care was taken to avoid selecting
methods and data that only confirmed already held opinions and perceptions. Thus, the practitioners view was cross-checked and inter-related with other empirical evidence in the two studies.

Lastly, a discussion about potential issues with the breadth and exploratory nature of my field studies is warranted. I have come to realize that my longitudinal, explorative studies probably were possible only within the frame of my long-term doctoral research. On the one hand, I see this as the equivalent of basic research in the natural sciences. It is important because it helps explore, describe and explain dynamics that are otherwise hard to study with the constraints of most research projects, where limited budgets and expectations of relatively short-term deliverables create a necessity for more concentrated research. On the other, while I see the importance of open-ended exploration for building a base for the research community, I also see that beyond doctoral research the more concentrated approach is often desirable as well as necessary, because it affects the practical contribution of the research.

Setting my argument within the interpretive and empirically grounded traditions that I have ascribed to, one can say that this is about the applicability of academic theory. Academic theories should be relevant to something or someone, should inform an activity of some sort. Albeit in the long-term and possibly via further research, I aim to inform practitioners, decision-makers, and designers with respect to the distributed software development domain. But this is not possible because of the statistical generalizability of my findings. Indeed, I have come to understand the idea of generalisability differently since the outset of my research. With qualitative, interpretive research efforts, we cannot claim that our findings and theories are generalisable in the sense that practitioners, decision-makers, or designers can simply use them as universal rules. This is a fundamental difference between research objectives in positivist and interpretivist traditions. In terms of the applicability of the research, this puts the role of scholars in a different light, as a theory can never truthfully be claimed to be generalizable to a new setting until it has empirically been tested within that setting (Lee and Baskerville 2003). With our expertise in research methods and intricate familiarity with the theories we are proponents for, we are also the most suitable candidates for shouldering the responsibility of bridging our findings on a particular case to other situations.
But in fast-moving domains such as distributed software development – and I think the same goes for most contemporary phenomena that are subjects in qualitative research – this can also be a difficult balance. From my experience with the two field studies, I have seen how this can manifest in actual field work practice. In the PyPy case study, the community members were actively encouraging researchers attention; not seeking direct feedback, but as part of the project goals and public image. But the situation was notably different in the Lambda study. Here it was quite obvious that my research moved much too slow compared to the demands of daily practice. My experience is that practitioners may expect instant feedback which is hard to provide, and they often consider the aspects highlighted in the research as unimportant in the spur of the moment. Efforts at “making the familiar unfamiliar” seem most likely to summon the interest of practitioners once in its finished, written-up state.

I think that this is an inherent property of my approach that affects the practical contributions of the thesis. While making a significant academic contribution, the exploratory approach weakened the direct applicability of the research somewhat. However, I also think that this is an opportunity for future research (such as that discussed in section 8.4). Now that this body of material has been produced and analysed, the work on actually applying it to specific problems can ensue.

8.6 Personal reflections

I would like to conclude this thesis with some personal remarks. I have labelled them as personal reflections as they are not intended to represent additional findings. Rather, they are an informal commentary on the contents of the thesis. I believe they are important, however, in providing an understanding for what I have strived to achieve and implications beyond the particular topic of my studies.

8.6.1 On the concept of knowledge and information systems design

My thesis is a contribution to the discipline of information systems. As I explained in the introductory chapter, I view distributed software development as a matter of groups performing systems development in real life contexts. An overarching purpose of my work is therefore to provide empirically-grounded material that may inform the design of information systems in supporting distributed software development efforts. The
Chapter 8 • Conclusions and reflections

class concept of knowledge is, for me, useful because it intuitively puts forward different aspects of the material and its meaning than if speaking only of how to provide information to people for various purposes. Walsham expressed this point of view as that “knowledge resides in human beings, not in computer systems, and communication is a complex process of human sense reading and sense giving, not the simple transfer of knowledge from one person to another” (Walsham 2002 pp. 267). I believe Brown and Duguid (2002) articulated this perspective very well in their book The Social Life of Information, where they made three key points about what a shift from talking about information to talking about knowledge entails. I paraphrase them here:

Firstly, in contrast to information, knowledge entails a knower. This puts our attention on people and what they are doing, rather than on, say, structure, storage and access (not meaning that these are unimportant, just that there are other equally important perspectives as well). Secondly, it also implies the understanding of a subject. You may have access to an unlimited amount of information, yet still not understand how to perform your task. But if you know how to do it, then you are by definition able to perform it. So discussing what is required for someone to know a task is another way of discussing exactly what resources and preparation they need. And thirdly, the acquisition or sharing of knowledge is (again intuitively) different from the acquisition or sharing of information. In fact, it is about learning rather than sharing, which implies a process and - assuming a practice-based perspective on knowledge like I have done in my thesis - uplifts the social interactions which underlie it.

So how, then, do my findings relate to the design of information systems? They show the concrete dynamics related specifically to what software developers do, what they need to know, and how this evolves through practice. But I have not striven to derive specific guidelines or heuristics or requirements. As argued by Dourish (2006), I believe that there is value in ethnographic-style investigations beyond treating them as methods for deriving specific requirements. My work has been an empirical contribution to the discourse on knowledge-related issues in distributed software development and I leave to it future research to make specific design decisions. But what I am doing, is putting forth one potential process that is important if we are to support in particular the continual maintenance of practical knowledge in the face of ever-evolving circumstances: the purposeful adaptation of practice, in which two
important concerns are anticipation and alignment.

The insights embedded in my research with respect to information systems design approaches align with the community approaches to knowledge management systems put forward by e.g. Bobrow & Whalen (Bobrow and Whalen 2002) and Erickson & Kellog (Erickson and Kellogg 2003). I also believe that my argument about supporting “reflective software developer groups” echoes those since long made in the field of Human-Computer Interaction about understanding users as active agents (Bannon 1991). Of course, it also has to be taken into consideration that these perspectives are relatively widespread today. The general design paradigm in the area of commercial CASE tools seems to be all about providing collaboration platforms. For instance, the Accord platform (a pseudonym) that figured in my field studies is marketed specifically as supporting communities of software developers. What I believe my findings are adding, though, are these specific descriptions of how knowledge is developed in distributed software development practice.

8.6.2 What is “Global Software Development”?
I have come a long way since I began working on the socGSD project in July 2006. Both in terms of personal development – though that's another story - and in my understanding of what “global software development” means. When I began reading up on the subject prior to joining the project, I accepted the view which was apparent to me in the literature. Global software development was a “phenomenon”. It was something distinctly new. Companies were looking to save money, while also gaining access to a wider pool of competence, by tapping into a global work force. In addition to Cooperation, Coordination, and Communication, such software projects were also hampered by issues of Distribution. This created entirely new problems, such as National Cultural Differences. Not only that, but Distribution also exacerbated other issues. Because of the distance there was no Informal Communication, so Coordination, Awareness, and Knowledge Transfer suddenly became even more problematic.

The socGSD project, however, provided a different frame for understanding distributed software development. Initiated by my supervisor Prof. Liam Bannon, the project focus was put on CSCW-inspired issues, looking at the global from a local point of view in terms of collaborative work practices. Ethnographically-inspired approaches were
favoured, and as I then embarked on my own foray into distributed software development with my field studies, I began to discern more clearly what this perspective meant. When myself and my colleagues in socGSD met the PyPy people in August 2006, they were more interested in talking about their sprint-driven development methodology in practical terms – what it allowed them to achieve in terms of collaborative software development – than as a way of “alleviating distance”. Sure, they were aware of being a dispersed community, but for them this was a collaborative challenge, one they addressed with the usual tools of open source projects (webpages, mailing lists, chat channels, version control systems, etc.) in combination with sprints. Later on, when I joined the team working in a global organisation, I was met by slight bewilderment when I began talking to them about what it was like to work in a “global software development project”. For them, it was just part of the job, having to coordinate and communicate with people in teams across the globe.

In the meantime, the more I read in the broader field, the more I began to ponder what was actually new in all this. Certainly, coordination may require a lot of planning and effort in distributed projects (as shown by Grinter, Herbsleb et al. 1999; Herbsleb, Mockus et al. 2000; Herbsleb, Mockus et al. 2001), but large-scale software development has always been very complex collaborative undertaking (as shown by Curtis, Krasner et al. 1988; Brooks 1995; Kraut and Streeter 1995). Was it not just a matter of scale? And cultural issues in software development were discussed without mentioning “global” distribution in particular (see especially Tellioglu and Wagner 1999), albeit not in terms of national culture - which, on the other hand, has been a major topic within strands of social science for a very long time, particularly in sociology and anthropology, as well as in globalization discourses in general (e.g. see Walsham 2001). Outsourcing has been a topic in information systems research for a long time (not to mention economics and business research). When did we pass the threshold where this became the phenomenon “Global Software Development”? This is the same issue as for technological definitions of the “Information Society” (see Webster 2006) - at what point, exactly, do we have enough information technology to be an information society? Additionally, as I was attending major conferences in the area, I was getting a feeling that the issues being discussed to a large extent were the same – or at least very closely related – to ones discussed for a long time in various computer
science disciplines (IS, CSCW, HCI, etc.), just put under a different label.

Finally, one particular reading made a great impression on me with respect to this pondering of mine: Erik S. Raymond's “The Cathedral and the Bazaar” (Raymond 2001).44 Albeit not an academic research work, this is a very insightful book for anyone interested in software development as an activity, both practitioners and spectators. What I particularly like is how he writes about the issues of open source software development in perfectly sensible ways, as something people are actually doing. But it was the chapter “A Brief History of Hackerdom” that especially struck me. Here was a history of some of the most interesting and cardinal software development efforts (Unix; BSD; Linux; ARPAnet; Internet; Usenet; World Wide Web; etc.) and movements (the hacker culture originating in, for example, MIT’s Artificial Intelligence Laboratory and the Computer Systems Research Group at University of California; the Free Software Foundation; etc.) of the 70’s, 80’s and 90’s. I was fascinated by the realisation that collaboration and exchange across distance via computer-based communication networks always has been pervasive in the software development domain (which maybe also tells us something about the tunnel-view perspective of those of us who entered the computer domain during mainstream boom of the late 90’s).

As scholars, it is our privilege to be given time to dissect activities and social interactions to an extent way beyond what any practitioner would or should. However, as such, we must also consider very carefully what it is that we are doing. We are, in fact, writing about other peoples everyday activities. In my work, I have not “discovered” anything new, because every single issue that I have written about is something thousands of people cope with every day. I have even agonised a great deal over using the word “findings” in my work, because I felt that it was somehow demeaning to the people I have studied. After all, they have vastly more experience with the issues analysed than I have.

Yet, I have done something that they could not have. I have spent the last 4 years wrestling with how to describe and situate these issues - reading, thinking, and writing; re-reading, re-thinking, and re-writing; over and over. They have since long moved on. My analysis has an entirely different scope. There are merits in both perspectives. They

---

44 In true open source spirit, the book itself is maintained online: http://catb.org/esr/writings/homesteading/ (Accessed: 2010-02-19)
are experts in their domain, and can cope with and innovate practice. I can provide an additional analysis of how they do this, one rooted in empirical material and propped up with academic concepts. “Making the familiar unfamiliar”, as it were, distilling the insights embedded in what they do. I nourish the hope that such an effort might help practitioners in their innovating by extracting the essential elements or providing a different perspective on what they're doing. Perhaps it may communicate insights to other practitioners who might find it useful for their work. Perhaps it will kindle and inform the design of future information systems and other tools to support practice.

But this, I believe, is the crux: when we talk about “Global Software Development”, we alienate many practitioners, because they – in my experience at least - think we are talking about something else than what they work with, about someone else's problems or perhaps something purely academic. One of the main arguments of my thesis has been that distribution is a mundane issue. It is a problem of what software developers actually do, what concerns they have, what resources they utilize, and so on. Personally, I believe scholars should focus on specific issues related to global collaboration – say, how sharing of knowledge happens between dispersed partners – *in particular contexts*, without reifying them, without reifying the phenomenon “Global Software Development”. Grander phenomena such as outsourcing, global organisations, inter-organisational collaboration, and open source communities are discourses of their own, separate from discourses focused on the actual activities within these contexts.

With my work, I have aspired to do just this by talking about constant change and the purposeful adaptation of practice. I have made it into a contribution to the discourse on knowledge-related issues, but my personal underlying interest has been to show distributed software development *in real life* seen through an academic lens. I am – originally and still – a software developer myself and know that what I spend most of my time doing is learning new things, be it, say, a new programming language, or a new API, or a new design pattern, or a new SDK, or whatever. My original interest in it as a subject has been just this, how software developers keep on learning in this rapidly and (let us be honest) often divergently evolving – but therefore extremely fascinating and engaging! – field.

I hope to have shown at least one possible side of this in my work and that, by making it empirically grounded - talking about what I have seen software developers actually *do* -
I have avoided reifying some high-level abstraction of what the work is about. Particularly, with the *purposeful* adaptation of practice, I hope to have given all software developers out there their due recognition as belonging to an extremely reflective and insightful breed. They are not robots, or cogwheels, or processors, but enthusiastic people who are well aware of what they are doing and the surrounding field in which they participate. It is by writing about them as such, and what they hence do, that I believe we as scholars can make a significant contribution.
Bibliography


Herbsleb, J. D., A. Mockus, et al. (2000). Distance, Dependencies, and Delay in a Global Collaboration. ACM Computer-supported cooperative work.


globalisation of software development: case experience from Britain and India."
Information and Organization 11: 25-43.


Nicolini, D., S. Gherardi, et al. (2003). Knowing in Organizations: A Practice-based

Companies Create the Dynamics of Innovation. New York, Oxford University Press.

Orlikowski, W. J. (1992). "The Duality of Technology: Rethinking the Concept of


Orlikowski, W. J. (2002). "Knowing in Practice: Enacting a Collective Capability in

Research on Information Technology and Research on Organizations Learn From Each
Other?" MIS Quarterly 25(2): 145-165.

New York, IRL Press.


Appendix A: Distributed software development – research landscape and key issues

Introduction
The purpose of this appendix is to provide a brief overview of research on the subject of distributed software development, so as to map the research landscape across several disciplines and highlight what main issues have been investigated. The content is based on a literature review that was undertaken within the socGSD project (see chapter 3 for more details on the project). It was a collaborative effort led by myself and what we argued was that there is a trend towards emphasising the situated activities through which software development is achieved in practice.

The research landscape - an interdisciplinary interest
Studies of the difficulties of globally distributed software development are frequent in software engineering forums (e.g. Grinter, Herbsleb et al. 1999; Herbsleb, Mockus et al. 2001; Ehrlich and Chang 2006; Hsieh 2006; Milewski 2007). As a field, software engineering is about providing a framework of development processes and techniques for building software with higher quality (Boehm 2006). From this perspective, the main concern has been how the collaborative work processes of distributed development projects can be managed.

Research on this topic is also common in information system forums (e.g. Sahay, Nicholson et al. 2003; Jarvenpaa, Shaw et al. 2004; Sarker and Sahay 2004; Imsland and Sahay 2005; Kotlarsky and Oshri 2005). Information systems is the study of how information and communication technology is used within organisations and the implications thereof (Avgerou 2000). Such a focus requires investigation of a wide range of issues. IS is an attempt to bridge the gap between engineering disciplines dedicated to developing technological capabilities and macro sciences such as sociology and economics that focus explaining both local sociocultural settings and broader societal and cultural contexts. This is reflected in the research on distributed software development in this area, as it tends to assume organisational and social perspectives.

The topic has also attracted a lot of attention in more specialized discourses such as management information systems (MIS) (e.g. Tractinsky and Jarvenpaa 1995; Carmel and Agarwal 2002), requirements engineering (e.g. Damian, Chisan et al. 2003;
Damian, Izquierdo et al. 2007), software processes (e.g. Gopal, Mukhopadhyay et al. 2002; Priklanicki, Audy et al. 2003), and computer-supported cooperative work (e.g. Herbsleb, Mockus et al. 2000; Cubranic, Murphy et al. 2004; DeSouza, Redmiles et al. 2004; Ducheneaut 2005; Fitzpatrick, Marshall et al. 2006; Halverson, Ellis et al. 2006). CSCW is particularly relevant in the context of this thesis, since I assume a methodology which is akin to approaches in this discipline. The focus of CSCW is on the issue of cooperative work and how it can be supported with computer-based tools (Bannon and Schmidt 1989; Schmidt and Bannon 1992). Correspondingly, the phenomenon of distributed software development is treated as a matter of computer-mediated collaboration across distance, but focus is often on actual, social practice through which the work is achieved.

Collaboration across distance – enabled by information technology and software tools - is also investigated much in areas outside the computer sciences, particularly organisation studies (e.g. Jarvenpaa and Leidner 1999; Maznevski and Chudoba 2000; Cramton 2001), and management and business studies (e.g. Ghemawat 2001; Kavanagh and Kelly 2002; Evaristo, Scudder et al. 2004). There is much similarity and cross-pollination between this body of research and research focused more specifically on distributed software development work. The perspective is related to that of information systems research, with organisational and social issues in the forefront. We also find research on distributed software development in forums on the borderline between organisation studies and information systems (e.g. Nicholson and Sahay 2001; Nicholson and Sahay 2004).

**Issues of distributed software development - towards an emphasis on situated activities**

The fact that economics is a major driving force behind the distribution of software projects is something that most researchers recognize. This makes it an important aspect of the work itself and economic incentives can be seen as one background factor that affects the work (Carmel and Agarwal 2002; Carmel and Tjia 2005). But the most prominent issues that are unearthed in research on distributed software development are related to coordination and communication. Whilst these have long been identified as key challenges in managing the complexity of software development projects, research indicates that the impact is more pronounced in a distributed environment (Grinter, Herbsleb et al. 1999; Herbsleb, Mockus et al. 2000; Herbsleb, Mockus et al. 2001).
Culture has been identified as yet another issue exacerbated by global distribution of software development (Priklanicki, Audy et al. 2003; Krishna, Sahay et al. 2004). While the aspect that most often is given explicit attention is that of national culture (Damian and Zowghi 2003; Hsieh 2006), there has also been a growing understanding of culture being a more complex dimension than that. For instance, Damian et al. reveal how different organisational cultures manifested in the way team processes are followed and the use of tools, and is an important factor in how awareness of change is promoted in a distributed project (Damian, Izquierdo et al. 2007). This carries with it the idea that culture as a concept also involves organisational culture, language, politics, individual motivations, work ethics, and so on. This notion is further elaborated by Sahay et al. whilst looking at cross-cultural communication issues between Japanese and Indian software firms (Sahay, Nicholson et al. 2003). They point to the fact that cross-cultural issues in distributed software projects are magnified by the abstract and knowledge-intensive nature of software development work, but specifically emphasise the problem of relying on static formulations of culture and ignoring the processes by which cultures are constituted and maintained. They focus on how the actors themselves act based on perceptions of other actors that sometimes equates national stereotypes with culture, but emphasise how this perception is continually socially constructed.

The importance of social relationships in the successful accomplishment of global collaboration is also evident. We see this mostly in research focused on virtual team dynamics, where studies of trust (Jarvenpaa and Leidner 1999; Jarvenpaa, Shaw et al. 2004), social ties (Kotlarsky and Oshri 2005), and shared mental models (Espinosa, Kraut et al. 2002) illustrate the impact of rapport. Trust in virtual teams is a concept that has received much attention within the organisation studies area. Interestingly, in this line of inquiry, there is a growing recognition that trust manifests itself and affects virtual teams differently in different situations (Jarvenpaa, Shaw et al. 2004). Jarvenpaa et al. are investigating context in the sense of project stages, yet it is an indication that the context in which people are acting is very important to account for when trying to understand the dynamics of the work in distributed settings.

This is even more apparent when Kavanagh and Kelly argue for a conceptual understanding of communication in distributed projects “as a collective sensemaking activity occurring within and between temporally and spatially situated communities of practice” (Kavanagh and Kelly 2002). In their case study, they show how the actors
activities are grounded in situated personal relationships. Similarly, in a study of coordination practices in distributed software development projects, Boden et al. conclude that these are intricately embedded in informal communication (Boden, Nett et al. 2007).

The point that these studies seem to converge on is how distributed software development as an activity is embedded in many different specific local contexts. We can see how, in the words of Tellioglū & Wagner, “actual practice depends on context (of people, knowledge, and the nature of the task) and is cultured (in the sense of being shaped by beliefs, commitments, styles, and power relations)” (Tellioglu and Wagner 1999). From this we can conclude that if we are to gain a useful understanding of distributed software development as a phenomenon, we need to complement organisation and process perspectives with studies of the situated, social activities that accomplish the software development in practice.
And that's probably how it happened...