STRATEGIZING AND EVALUATING THE ONBOARDING OF SOFTWARE DEVELOPERS IN LARGE-SCALE GLOBALLY DISTRIBUTED LEGACY PROJECTS

Ricardo Britto

Blekinge Institute of Technology
Doctoral Dissertation Series No. 2017:09
Department of Software Engineering
Strategizing and Evaluating the Onboarding of Software Developers in Large-Scale Globally Distributed Legacy Projects

Ricardo Britto
Strategizing and Evaluating the Onboarding of Software Developers in Large-Scale Globally Distributed Legacy Projects

Ricardo Britto

Doctoral Dissertation in Software Engineering

Department of Software Engineering
Blekinge Institute of Technology
SWEDEN
To my wife, daughter, parents and siblings.

“I fear not the man who has practiced 10,000 kicks once, but I fear the man who has practiced one kick 10,000 times.”

–Bruce Lee

“In God we trust, all others must bring data”

–W. Edward Dennings

“There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don’t know. But there are also unknown unknowns. There are things we don’t know we don’t know.”

–Donald Rumsfeld
ABSTRACT

**Background:** Recruitment and onboarding of software developers are essential steps in software development undertakings. The need for adding new people is often associated with large-scale long-living projects and globally distributed projects. The formers are challenging because they may contain large amounts of legacy (and often complex) code (legacy projects). The latters are challenging, because the inability to find sufficient resources in-house may lead to onboarding people at a distance, and often in many distinct sites. While onboarding is of great importance for companies, there is little research about the challenges and implications associated with onboarding software developers and teams in large-scale globally distributed projects with large amounts of legacy code. Furthermore, no study has proposed any systematic approaches to support the design of onboarding strategies and evaluation of onboarding results in the aforementioned context.

**Objective:** The aim of this thesis is two-fold: i) identify the challenges and implications associated with onboarding software developers and teams in large-scale globally distributed legacy projects; and ii) propose solutions to support the design of onboarding strategies and evaluation of onboarding results in large-scale globally distributed legacy projects.

**Method:** In this thesis, we employed literature review, case study, and business process modeling. The main case investigated in this thesis is the development of a legacy telecommunication software product in Ericsson.

**Results:** The results show that the performance (productivity, autonomy, and lead time) of new developers/teams onboarded in remote locations in large-scale distributed legacy projects is much lower than the performance of mature teams. This suggests that new teams have a considerable performance gap to overcome. Furthermore, we learned that onboarding problems can be amplified by the following challenges: complexity of the product and technology stack, distance to the main source of product knowledge, lack of team stability, training expectation misalignment, and lack of formalism and control over onboarding strategies employed in different sites of globally distributed projects. To help companies addressing the challenges we identified in this thesis, we propose a process to support the design of onboarding strategies and the evaluation of onboarding results.

**Conclusions:** The results show that scale, distribution and complex legacy code may make onboarding more difficult and demand longer periods of time for new developers and teams to achieve high performance. This
means that onboarding in large-scale globally distributed legacy projects must be planned well ahead and companies must be prepared to provide extended periods of mentoring by expensive and scarce resources, such as software architects. Failure to foresee and plan such resources may result in effort estimates on one hand, and unavailability of mentors on another, if not planned in advance. The process put forward herein can help companies to deal with the aforementioned problems through more systematic, effective and repeatable onboarding strategies.
ACKNOWLEDGMENTS

Although a Ph.D. thesis often seems as a lonely journey, I soon realized during the course of my odyssey that this is not entirely true. First, I would like to express my gratitude to my supervisors Prof. Darja Šmite and Prof. Jürgen Börstler, and also to Prof. Claes Wohlin and Prof. Emilia Mendes for their support, guidance and fruitful collaboration that was mandatory for the success of this thesis.

Furthermore, I would like also to express my gratitude to Ericsson, which allowed me to conduct my research and put forward my ideas in a real world case. More specifically, I would like to thank Lars-Ola Damm for all the support, ideas and inspiring meetings that helped me to mold the main contributions of this thesis. At the same time, I could not forget to say thank you to all the software architects, developers and project managers from Ericsson who provided their time and knowledge that helped to make this thesis possible.

I also would like to thank all my friends and colleagues at SERL Sweden, who also contributed to this thesis by providing a pleasant work environment. In particular, I would like to express my gratitude to Muhammad Usman for his collaboration on papers and for all the great times after work.

My deepest gratitude goes to my family, especially my wife Annyele Moura (love of my life), who is always there, especially in the hardest moments. To my daughter Maria-Luisa, who is the most valuable gift I received from God, and who every day gives light to my life. I also would like to thank my parents José Antônio and Teresa, my siblings Juliana, Hugo Leonardo and David, and all my friends from Sweden and Brazil (fortunately, the list is too long to include here).

This work was funded by the Swedish Knowledge Foundation (KKS), Federal University of Piauí (UFPI), National Council for Scientific and Technological Development (CNPq) and National Institute of Science and Technology for Software Engineering (INES), for which I also would like to express my gratitude.

Karlskrona, November 03, 2017
This compilation thesis includes the following six papers:


**Paper 2 (P2, Chapter 3)** – Ricardo Britto, Darja Šmite and Lars-Ola Damm. ‘Experiences from measuring learning and performance in large-scale distributed software development’. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Ciudad Real, Spain, Pages 1-6, Article 17, 2016 (Best Paper Award).


**CONTRIBUTION STATEMENT**

Ricardo Britto was the lead author of P1–P5. The responsibility as the lead author was to lead design, conduct data collection/analysis and led the write-up of each study. In P6, Ricardo Britto led the data collection, while
Muhammad Usman led the data analysis. The main responsibility regarding design and write-up were equally shared by both authors. A more detailed description of each author’s contribution is as follows:

- **P1** – Ricardo Britto was responsible for identifying related work, research design and executing the process to extend Šmite et al.’s taxonomy \[214\], validation, and being the main responsible for writing the manuscript. Claes Wohlin conceptualized the main idea of the paper. He also provided expert knowledge on GSE and knowledge about the original taxonomy. Finally, he also contributed to the research design, feedback and editorial work on the manuscript. Emilia Mendes contributed to the research design, feedback, and editorial work on the manuscript. All authors read and approved the final manuscript.

- **P2** – Ricardo Britto was responsible for conceptualizing the main idea of the paper, research designing, identifying related work, collecting and analyzing data, and being the main responsible for writing the manuscript. Darja Šmite contributed to the conceptualization of the main idea of the paper, research designing, write-up, and editorial work. Lars-Ola Damm contributed to the data analysis and provided feedback. All authors read and approved the final manuscript.

- **P3** – Ricardo Britto was responsible for conceptualizing the main idea of the paper, research design, identifying related work, collecting and analyzing data, and being the main responsible for writing the manuscript. Darja Šmite contributed to the conceptualization of the main idea of the paper, research design, write-up, and editorial work. Lars-Ola Damm contributed to the data collection and analysis, write-up, and provided feedback. All authors read and approved the final manuscript.

- **P4** – Ricardo Britto was responsible for conceptualizing the main idea of the paper, research design, identifying related work, collecting and analyzing data, and being the main responsible for writing the manuscript. Darja Šmite contributed to the research design and editorial work, and provided feedback. Lars-Ola Damm contributed to the data collection and analysis, and provided feedback. Jürgen Börstler contributed to the editorial work, and provided feedback. All authors read and approved the final manuscript.

- **P5** – Ricardo Britto was responsible for conceptualizing the main idea of the paper, research design, identifying related work, collecting and analyzing data, and being the main responsible for writing the manuscript. Daniela Cruzes contributed to the data collection and analysis, write-up, and editorial work. Darja Šmite contributed to the data analysis, write-up, and editorial work. Aivars Šablis contributed to the data collection and analysis, and write-up. All authors read and approved the final manuscript.
• P6 – Ricardo Britto was responsible for conceptualizing the main idea of the paper, research design, identification of related work, data collection, write-up and editorial work. Muhammad Usman was also responsible for conceptualizing the main idea of the paper, research design, data analysis, write-up and editorial work. Lars-Ola Damm contributed to the data collection, and provided feedback. Jürgen Börstler contributed to the editorial work, and provided feedback. All authors read and approved the final manuscript.

RELATED PAPERS NOT INCLUDED IN THIS THESIS


Global Software Engineering (ICGSE), Shanghai, China, Pages 18-21, 2014.


**OTHER PAPERS NOT INCLUDED IN THIS THESIS**


**Paper 20** – Dennis Silva, Ricardo Rabelo, Matheus Campanha, Pedro Santos Neto, Pedro Oliveira and Ricardo Britto. ‘A hybrid approach for test


# CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ix</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1 Problem outline</td>
<td>1</td>
</tr>
<tr>
<td>2 Research context</td>
<td>2</td>
</tr>
<tr>
<td>2.1 What are we studying?</td>
<td>2</td>
</tr>
<tr>
<td>2.2 Why are we interested in it?</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Why should this be interesting to others?</td>
<td>4</td>
</tr>
<tr>
<td>2.4 Research questions</td>
<td>4</td>
</tr>
<tr>
<td>3 Chapter overview</td>
<td>4</td>
</tr>
<tr>
<td>4 Background</td>
<td>5</td>
</tr>
<tr>
<td>4.1 Global software engineering</td>
<td>5</td>
</tr>
<tr>
<td>4.2 Onboarding</td>
<td>7</td>
</tr>
<tr>
<td>4.3 Learning in the context of the onboarding software developers and teams</td>
<td>12</td>
</tr>
<tr>
<td>5 Related work</td>
<td>14</td>
</tr>
<tr>
<td>6 Research gaps</td>
<td>16</td>
</tr>
<tr>
<td>7 Research design and methods</td>
<td>16</td>
</tr>
<tr>
<td>7.1 Case study</td>
<td>17</td>
</tr>
<tr>
<td>7.2 Cases in this thesis</td>
<td>18</td>
</tr>
<tr>
<td>7.3 Business process modeling</td>
<td>21</td>
</tr>
<tr>
<td>7.4 Literature review</td>
<td>22</td>
</tr>
<tr>
<td>8 Contributions and summary of included research papers</td>
<td>24</td>
</tr>
<tr>
<td>8.1 Chapter 2 (P1) – An extended global software engineering taxonomy</td>
<td>24</td>
</tr>
<tr>
<td>8.2 Chapter 3 (P2) – Experiences from measuring learning and performance in large-scale distributed software development</td>
<td>25</td>
</tr>
<tr>
<td>8.3 Chapter 4 (P3) – Software architects in large-scale distributed projects: An Ericsson case</td>
<td>25</td>
</tr>
<tr>
<td>8.4 Chapter 5 (P4) – Learning and performance evolution of immature remote teams in large-scale software projects: An industrial case study</td>
<td>26</td>
</tr>
<tr>
<td>8.5 Chapter 6 (P5) – Onboarding software developers and teams in three globally distributed legacy projects: A multi-case study</td>
<td>27</td>
</tr>
<tr>
<td>8.6 Chapter 7 (P6) – Effort estimation in large-scale distributed agile software development: An industrial case study</td>
<td>28</td>
</tr>
<tr>
<td>Page</td>
<td>Section</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>9</td>
<td>The proposed process</td>
</tr>
<tr>
<td>9.1</td>
<td>Define context</td>
</tr>
<tr>
<td>9.2</td>
<td>Strategize onboarding undertaking</td>
</tr>
<tr>
<td>9.3</td>
<td>Evaluate onboarding results</td>
</tr>
<tr>
<td>9.4</td>
<td>Identify and address issues</td>
</tr>
<tr>
<td>9.5</td>
<td>Using the process</td>
</tr>
<tr>
<td>10</td>
<td>Discussion</td>
</tr>
<tr>
<td>10.1</td>
<td>Validity threats</td>
</tr>
<tr>
<td>11</td>
<td>Conclusions, implications and future directions</td>
</tr>
<tr>
<td>2</td>
<td><strong>AN EXTENDED GLOBAL SOFTWARE ENGINEERING TAXONOMY</strong></td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>1.1</td>
<td>Context</td>
</tr>
<tr>
<td>1.2</td>
<td>Problem outline</td>
</tr>
<tr>
<td>1.3</td>
<td>Objective</td>
</tr>
<tr>
<td>1.4</td>
<td>Contribution</td>
</tr>
<tr>
<td>1.5</td>
<td>Outline</td>
</tr>
<tr>
<td>2</td>
<td>Background and literature review</td>
</tr>
<tr>
<td>2.1</td>
<td>Smite et al.’s GSE taxonomy</td>
</tr>
<tr>
<td>2.2</td>
<td>Additional related work</td>
</tr>
<tr>
<td>2.3</td>
<td>Research gaps</td>
</tr>
<tr>
<td>3</td>
<td>Research design and methodology</td>
</tr>
<tr>
<td>4</td>
<td>The extended GSE taxonomy</td>
</tr>
<tr>
<td>4.1</td>
<td>GSE</td>
</tr>
<tr>
<td>4.2</td>
<td>Setting</td>
</tr>
<tr>
<td>4.3</td>
<td>Software process dimensions</td>
</tr>
<tr>
<td>4.4</td>
<td>Cultural factors</td>
</tr>
<tr>
<td>4.5</td>
<td>Language distance</td>
</tr>
<tr>
<td>4.6</td>
<td>Communication model</td>
</tr>
<tr>
<td>5</td>
<td>Validation</td>
</tr>
<tr>
<td>5.1</td>
<td>Demonstration of utility</td>
</tr>
<tr>
<td>5.2</td>
<td>Benchmarking</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
</tr>
<tr>
<td>7</td>
<td>Limitations</td>
</tr>
<tr>
<td>8</td>
<td>Conclusions and future work</td>
</tr>
<tr>
<td>3</td>
<td>EXPERIENCES FROM MEASURING LEARNING AND PERFORMANCE IN LARGE-SCALE DISTRIBUTED SOFTWARE DEVELOPMENT</td>
</tr>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>Background</td>
</tr>
<tr>
<td>2.1</td>
<td>Performance</td>
</tr>
<tr>
<td>2.2</td>
<td>Learning</td>
</tr>
<tr>
<td>2.3</td>
<td>Knowledge, skill and competence</td>
</tr>
<tr>
<td>3</td>
<td>The case and unit of analysis</td>
</tr>
</tbody>
</table>
4 Measurement approach .......................................................... 82
  4.1 Data preparation ............................................................. 84
  4.2 Data analysis ................................................................. 85
5 Examples and lessons learned .................................................. 85
6 Reflections and discussion ....................................................... 90
7 Validity threats ................................................................. 91
8 Conclusions and future work .................................................... 92

4 SOFTWARE ARCHITECTS IN LARGE-SCALE DISTRIBUTED PROJECTS:
AN ERICSSON CASE ................................................................. 93
1 Introduction ................................................................. 93
2 Research design ............................................................. 94
3 Results ............................................................................ 95
  3.1 RQ1 – Roles, responsibilities and organization ............ 95
  3.2 RQ2 – The influence of team maturity, scale and dis-
  tribution on the architects’ role ........................................ 97
4 Discussion .................................................................. 101
5 Validity threats ............................................................. 103
6 Conclusions and future work ................................................. 104

5 LEARNING AND PERFORMANCE EVOLUTION OF IMMATURE
REMOTE TEAMS IN LARGE-SCALE SOFTWARE PROJECTS: AN
INDUSTRIAL CASE STUDY ................................................................. 107
1 Introduction ................................................................. 107
2 Background and related work ............................................ 109
  2.1 Learning and performance ............................................ 109
  2.2 Performance and learning in software engineering .... 111
  2.3 Performance and learning in global software engi-
  neering ......................................................................... 113
3 Research design ............................................................. 115
  3.1 The case and unit of analysis ........................................ 115
  3.2 Variables .................................................................. 118
  3.3 Data collection ............................................................ 119
  3.4 Data preparation .......................................................... 125
  3.5 Data analysis ............................................................... 125
4 Results ........................................................................... 126
  4.1 RQ1 – Performance evolution of immature remote
  teams ........................................................................ 126
  4.2 RQ2 – Performance gap between immature remote
  teams and mature teams ............................................... 137
5 Discussion ................................................................ 140
  5.1 Implications for research ............................................. 147
  5.2 Implications for practice .............................................. 147
6 Threats to validity ........................................................... 149
7 Conclusions and future work ................................................. 152
## ONBOARDING SOFTWARE DEVELOPERS AND TEAMS IN THREE GLOBALLY DISTRIBUTED LEGACY PROJECTS: A MULTI-CASE STUDY

1. Introduction .......................................................... 155
2. Background and related work ........................................ 156
   2.1 Onboarding .................................................. 157
   2.2 Van Maanen and Shein’s model ............................... 157
   2.3 Jones’ model ................................................ 158
   2.4 Bauer’s model ............................................... 158
   2.5 Onboarding in the software development context .......... 162
3. Research design .................................................... 164
   3.1 Case description ............................................. 166
   3.2 Data collection ............................................. 166
   3.3 Data preparation and analysis ................................ 167
4. Results ................................................................. 169
   4.1 Case 1 ....................................................... 169
   4.2 Case 2 ....................................................... 174
   4.3 Case 3 ....................................................... 176
5. Discussion ............................................................. 179
   5.1 Formality level of onboarding strategies ..................... 179
   5.2 Onboarding challenges for globally distributed legacy projects ........................................ 181
   5.3 Recommendations and implications for practice .......... 182
6. Threats to validity .................................................. 184
7. Conclusions and future work ....................................... 186

## EFFORT ESTIMATION IN LARGE-SCALE DISTRIBUTED AGILE SOFTWARE DEVELOPMENT: AN INDUSTRIAL CASE STUDY

1. Introduction .......................................................... 189
2. Related work ........................................................ 190
3. Research methodology .............................................. 192
   3.1 The case and unit of analysis ................................ 192
   3.2 Data collection ............................................. 193
   3.3 Data preparation ............................................. 194
   3.4 Data analysis ................................................. 195
4. Results ................................................................. 196
   4.1 RQ1: How is effort estimation carried out? .................. 196
   4.2 RQ2: How accurate are the effort estimates? ............... 198
   4.3 RQ3: Which factors impact the accuracy of the effort estimates? ........................................... 201
   4.4 RQ4: How do the identified factors impact the accuracy of the effort estimates? ...................... 205
   4.5 Results from the interviews ................................... 211
5. Discussion ............................................................. 211
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Classification of the included research papers and sections according to the main aims of the thesis.</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Model to support the development of onboarding strategies (adapted from Bauer [9]).</td>
<td>13</td>
</tr>
<tr>
<td>1.3</td>
<td>The evolution of the sites involved with the development and maintenance of the product [30].</td>
<td>20</td>
</tr>
<tr>
<td>1.4</td>
<td>Process modeling.</td>
<td>23</td>
</tr>
<tr>
<td>1.5</td>
<td>The proposed process.</td>
<td>30</td>
</tr>
<tr>
<td>1.6</td>
<td>The authority matrix [27] showing maturity levels A–D.</td>
<td>36</td>
</tr>
<tr>
<td>1.7</td>
<td>Onboarding timeline for product customization teams in India [30].</td>
<td>38</td>
</tr>
<tr>
<td>1.8</td>
<td>Sites involved in the example.</td>
<td>38</td>
</tr>
<tr>
<td>2.1</td>
<td>The GSE taxonomy (Adapted from Smite et al. [214]).</td>
<td>50</td>
</tr>
<tr>
<td>2.2</td>
<td>The process employed to extend Smite et al.'s taxonomy.</td>
<td>54</td>
</tr>
<tr>
<td>2.3</td>
<td>Extended GSE taxonomy.</td>
<td>56</td>
</tr>
<tr>
<td>2.4</td>
<td>Setup of the classified projects.</td>
<td>65</td>
</tr>
<tr>
<td>3.1</td>
<td>Examples of learning curve (adapted from Anzanello and Fogliatto [4]).</td>
<td>79</td>
</tr>
<tr>
<td>3.2</td>
<td>Example of performance curve.</td>
<td>83</td>
</tr>
<tr>
<td>3.3</td>
<td>Data sources.</td>
<td>85</td>
</tr>
<tr>
<td>3.4</td>
<td>Productivity curve of Team 1.</td>
<td>86</td>
</tr>
<tr>
<td>3.5</td>
<td>Productivity curve of Team 2.</td>
<td>86</td>
</tr>
<tr>
<td>3.6</td>
<td>Productivity curve of Team 3.</td>
<td>87</td>
</tr>
<tr>
<td>4.1</td>
<td>The organization of Ericsson's software architects.</td>
<td>96</td>
</tr>
<tr>
<td>4.2</td>
<td>The authority matrix.</td>
<td>98</td>
</tr>
<tr>
<td>4.3</td>
<td>The involved teams.</td>
<td>99</td>
</tr>
<tr>
<td>4.4</td>
<td>The effort spent by product-level architects supporting the teams.</td>
<td>100</td>
</tr>
<tr>
<td>4.5</td>
<td>Amount of defects revealed by product-level architects and teams.</td>
<td>101</td>
</tr>
<tr>
<td>5.1</td>
<td>The impact of different factors on performance and related studies.</td>
<td>111</td>
</tr>
<tr>
<td>5.2</td>
<td>The evolution of the sites involved with the development and maintenance of the product.</td>
<td>116</td>
</tr>
<tr>
<td>5.3</td>
<td>Onboarding timeline for product customization (PC) teams in India.</td>
<td>118</td>
</tr>
</tbody>
</table>
Figure 5.4 Team productivity evolution for immature teams based on task completion date (24 tasks) .... 127
Figure 5.5 Boxplot of team productivity and distribution .... 130
Figure 5.6 Team productivity versus task developer ratio for immature teams (24 tasks) .... 131
Figure 5.7 Boxplot of team productivity and task developer ratio .... 132
Figure 5.8 Team autonomy evolution for immature teams based on task completion date (24 tasks) .... 133
Figure 5.9 Boxplot of team autonomy and distribution .... 135
Figure 5.10 Team autonomy versus task developer ratio for immature teams (24 tasks) .... 136
Figure 5.11 Boxplot of team autonomy and task developer ratio for immature teams (24 tasks) .... 138
Figure 5.12 Boxplot of team productivity and maturity .... 139
Figure 5.13 Boxplot of team autonomy regarding maturity .... 141
Figure 5.14 The authority matrix [27] .... 150
Figure 6.1 Model to support the development of onboarding strategies (adapted from Bauer [9]) .... 161
Figure 6.2 Summary of the onboarding functions in Case 1 .... 173
Figure 6.3 Summary of the onboarding functions in Case 2 .... 175
Figure 6.4 Summary of the onboarding functions for existing sites with local seniors in Case 3 .... 177
Figure 6.5 Summary of the onboarding functions for newly established remote sites in Case 3 .... 177
Figure 7.1 Effort estimation process (PC = Product customization) .... 196
Figure 7.2 BREbias of QEs of PCs arranged on actual effort .... 206
Figure 7.3 BREBias boxplots by size for QE .... 206
Figure 7.4 BREbias of AEs of PCs arranged on actual effort .... 207
Figure 7.5 BREBias boxplots by size for AE .... 208
Figure A.1 Interview guide 1 .... 219
Figure A.2 Interview guide 2 .... 220
Figure B.1 Histogram of residuals .... 222
Figure B.2 Normal PP Plot of regression standardized residuals .... 223
Figure B.3 Scatter plot standardized residuals and predicted values .... 224
Figure C.1 Histogram of residuals .... 226
Figure C.2 Normal PP Plot of regression standardized residuals .... 227
Figure C.3 Scatter plot standardized residuals and predicted values .... 228
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1</td>
<td>Sourcing main characteristics.</td>
<td>6</td>
</tr>
<tr>
<td>Table 1.2</td>
<td>Global distances impact on organizational unit processes (adapted from Ågerfalk et al. [3])</td>
<td>8</td>
</tr>
<tr>
<td>Table 1.3</td>
<td>Dimensions of the GSE taxonomy at the site level.</td>
<td>31</td>
</tr>
<tr>
<td>Table 1.4</td>
<td>Dimensions of the GSE taxonomy at the relationship-between-pair-of-sites level - Part I.</td>
<td>32</td>
</tr>
<tr>
<td>Table 1.5</td>
<td>Dimensions of the GSE taxonomy at the relationship-between-pair-of-sites level - Part II.</td>
<td>33</td>
</tr>
<tr>
<td>Table 1.6</td>
<td>Classification at the site level for the example.</td>
<td>39</td>
</tr>
<tr>
<td>Table 1.7</td>
<td>Classification at the relationship level for the example.</td>
<td>40</td>
</tr>
<tr>
<td>Table 2.1</td>
<td>Dimensions of Smite et al.’s taxonomy.</td>
<td>51</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Classification in the site level – part I.</td>
<td>66</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Classification in the site level – part II.</td>
<td>67</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>Classification in the relationship-between-pair-of-sites level – part I.</td>
<td>68</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>Classification in the relationship-between-pair-of-sites level – part II.</td>
<td>69</td>
</tr>
<tr>
<td>Table 2.6</td>
<td>Comparison between the extended taxonomy, the base taxonomy and Gumm’s taxonomy – part II.</td>
<td>71</td>
</tr>
<tr>
<td>Table 2.7</td>
<td>Comparison between the extended taxonomy, the base taxonomy and Gumm’s taxonomy – part II.</td>
<td>72</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Impact factors’ details.</td>
<td>112</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Distributed product customization tasks.</td>
<td>121</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Archival research data sources – part I.</td>
<td>122</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Archival research data sources – part II.</td>
<td>123</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Descriptive statistics of team productivity in relation to distribution.</td>
<td>129</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Descriptive statistics of team productivity in relation to task developer ratio.</td>
<td>130</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Descriptive statistics of team autonomy in relation to distribution.</td>
<td>134</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Descriptive statistics of team autonomy regarding task developer ratio.</td>
<td>137</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>Descriptive statistics of team productivity in relation to maturity.</td>
<td>138</td>
</tr>
<tr>
<td>Table 5.10</td>
<td>Descriptive statistics of team autonomy and maturity.</td>
<td>140</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Relationship between the related work and Bauer’s model.</td>
<td>165</td>
</tr>
</tbody>
</table>
Table 6.2 Details about the interviews. .................................... 168
Table 6.3 Results regarding recruitment. ............................... 170
Table 6.4 Results regarding orientation. ............................... 170
Table 6.5 Results regarding coaching and support. .................. 171
Table 6.6 Results regarding training. ................................... 171
Table 6.7 Results regarding support tools and processes. ......... 172
Table 6.8 Results regarding feedback. ................................ 172
Table 7.1 Description of semi-structured interviewee sample. .... 194
Table 7.2 Mapping RQs with the used data collection methods. . 194
Table 7.3 Accuracy distribution (overestimation) of the two estimates (QE and AE). ............................ 199
Table 7.4 Accuracy distribution (accurately) of the two estimates (QE and AE). .................................... 200
Table 7.5 Accuracy distribution (underestimation) of the two estimates (QE and AE). ............................. 200
Table 7.6 Mean and Median BRE and BREbias for both estimates (QE and AE). .................................. 200
Table 7.7 Testing the two estimates (QE and AE) for significant differences. ........................................ 201
Table 7.8 Mean and median actual effort for all 60 PCs in work hours grouped by PC size. ....................... 205
Table 7.9 Mean and median BREBias for both estimates (QE and AE) grouped by PC size. ....................... 208
Table 7.10 Mean and median BREBias for both estimates (QE and AE) grouped by PC priority. .................. 209
Table 7.11 Mean and median BREBias for both estimates (QE and AE) for immature and mature teams. ....... 210
Table 7.12 Mean and median BREBias for both estimates (QE and AE) for collocated and multi-site PCs. ...... 211
Table B.1 BREbias QE regression analysis: ANOVA. ................. 221
Table B.2 BREbias QE regression coefficients’ details. ............ 221
Table B.3 BREbias QE regression models’ summary. ............... 221
Table C.1 BREbias AE regression: ANOVA. .......................... 225
Table C.2 BREbias AE regression coefficients’ details. ............ 225
Table C.3 BREbias AE regression models’ summary. ............... 225
1 INTRODUCTION

1 PROBLEM OUTLINE

Software product development is a challenging task that many companies have to handle on a daily basis. One of the key steps of software development is related to recruiting and onboarding software developers [9, 71, 72]. Onboarding (also known as organizational socialization) is the process of supporting newcomers (e.g., software developers) regarding the job, social, and performance adjustments [9].

In the context of software development, many reasons may lead to frequent onboarding of new software developers and teams such as: i) to increase manpower; ii) to incorporate new people aiming at innovation; iii) to replace retired developers; iv) to replace developers that left the company; or v) to free up developers to other projects or business units within the same company. Effective onboarding strategies are essential to ensure that new developers will be able to timely acquire the attitudes, knowledge, and skills required to work effectively [9, 11, 37, 71, 72, 120]. Successful onboarding is related to high job satisfaction, high organizational commitment, high performance, high career effectiveness, low staff turnover, and low stress [9, 13, 138, 146, 182].

In some situations, software developers are onboarded in globally distributed projects (Global Software Engineering – GSE) [21, 49, 88, 172], which often involve a large number of people (large-scale projects1), have long life cycles and large amounts of (often complex) legacy code [206]. The combination of scale and distribution makes coordination and communication more difficult. More complicated communication and coordination, in addition to legacy code, make the learning and mentoring process of new developers (and consequently onboarding) more challenging [19, 43, 72, 212]. The onboarding of software developers, especially when they are immature and located in offshore locations, also may have conse-

---

1 Dikert et al. define large-scale software undertakings as the ones that involve at least 50 human resources – not necessarily only developers, but also other staff collaborating in software development – or at least six teams [57].
quences for software projects, such as bigger uncertainty regarding effort estimates [24, 200] and bigger labor costs [211].

While a lot of research has been carried out regarding onboarding of newcomers in different roles and professions [12, 82, 120, 136, 177], there is little research about the challenges and implications associated with the onboarding of software developers and teams in **large-scale globally distributed projects with large amounts of legacy code** (legacy projects). Furthermore, no study has proposed any systematic approach to support the development of onboarding **strategies** and **evaluation** of onboarding results\(^2\) in the aforementioned context. Existing literature suggests that formal and systematic onboarding strategies are more successful [9]. Thus, the identification of the associated challenges and consequently the development of a systematic approach has the potential to help companies to be more successful when carrying out onboarding undertakings in the aforementioned context.

## 2 Research Context

This thesis summarizes the findings of empirical investigations regarding onboarding in large-scale globally distributed legacy projects and was mainly driven by the needs of Ericsson\(^3\). The main aim of this thesis is two-fold: i) identify the challenges and implications associated with onboarding software developers and teams in large-scale globally distributed legacy projects; and ii) propose solutions to support the development of onboarding strategies and evaluation of onboarding results in the context of large-scale globally distributed legacy projects.

### 2.1 What are we studying?

In this thesis, we investigated the onboarding of software developers and teams in large-scale globally distributed legacy projects. Two achieve the aforementioned main aims, we conducted three investigations, which were reported in six research papers (P\(_1\) [28] – Chapter 2, P\(_2\) [26] – Chapter 3, P\(_3\) [27] – Chapter 4, P\(_4\) [30] – Chapter 5, P\(_5\) [29] – Chapter 6, and P\(_6\) [202] – Chapter 7). Furthermore, the thesis itself has novel contribution that helped to achieve the main goals, which are detailed in Sections 9 and 10. Figure 1.1 shows a classification of the research papers and sections according to the main aims of the thesis.

\(^2\) In this thesis, onboarding results mean the performance of onboarded developers and teams after some time.

\(^3\) www.ericsson.com
Figure 1.1: Classification of the included research papers and sections according to the main aims of the thesis.
2.2 Why are we interested in it?

The investigated organization (Ericsson) employs GSE practices to develop a diverse constellation of products, i.e. it is often the case that software developers and teams are onboarded in projects that involve several developers located in different countries. Thus, it was of special interest for Ericsson to: i) shed light on the challenges and implications associated with onboarding developers and teams in this context; ii) identify whether the strategies currently employed in the company were effective; and iii) address any identified issue related to their onboarding strategies.

2.3 Why should this be interesting to others?

There are many other companies that have challenges similar to the ones faced by Ericsson (e.g. increased number of trouble reports and longer lead time after transfers/onboarding of developers in remote locations [106]). Thus, solutions already in place and solutions proposed in the context of this thesis have the potential of being effective and applicable in other companies operating in a similar context. Furthermore, the environment wherein the research reported in this thesis took place was also fruitful in revealing new research opportunities.

2.4 Research questions

This thesis addresses the following research questions:

- **RQ1** - What are the challenges and implications of onboarding software developers and teams in large-scale globally distributed legacy projects?
- **RQ2** - How can companies systematically strategize and evaluate the onboarding of software developers and teams in large-scale globally distributed legacy projects?

RQ1 was mainly addressed by the investigations reported in P2–P6, while RQ2 was addressed as a combination of the solutions put forward in P1–P5.

3 Chapter overview

The remainder of this chapter is organized as follows: Section 4 contains relevant background, which together with a discussion of the related work (Section 5) and existing research gaps (Section 6), sets the context for the thesis and its contributions. Section 7 contains the research approach, research design and methods employed in this thesis. Section 8 presents the
novel contribution of this thesis and a summary of the research papers included in this thesis. In Section 9, we describe the process to support the development of onboarding strategies and evaluation of onboarding results. Section 10 contains a discussion of the results and associated validity threats. Finally, Section 11 presents conclusions, implications and future directions.

4 BACKGROUND

The research conducted and presented in this thesis mainly relates to three topics: i) global software engineering; ii) onboarding; and iii) learning in the context of the onboarding of software developers and teams. Therefore, in this section, we provide key definitions associated with these three topics.

4.1 Global software engineering

The main motivation of organizations for developing software in a globally distributed fashion (employing GSE practices) is to gain or maintain a competitive advantage in terms of cost, time to market, quality, flexibility, productivity and risk dilution in software development [88, 185]. Globally distributed projects always involve some kind of sourcing (some sort of external software development that involves at least two different sites), which has the following main characteristics: location, legal entity, geographic distance and temporal distance. These characteristics are further elaborated in Table 1.1.

Although developing software in a globally distributed manner enables some benefits as mentioned above, globally distributed projects are associated with temporal, geographical and socio-cultural distances (global distances) [3], which make coordination, control and communication more challenging in globally distributed projects [39]. Temporal distance measures the time difference between the actors of two different organizational units. Temporal distance is especially caused by time zone differences. Geographical distance measures the effort to enable the actors of two different organizational units to interact personally (on site) and is captured by considering the effort to travel between organizational units. Socio-cultural distance measures the effort to enable the actors of two different organizational units to gain mutual understanding. The mutual understanding is about values/practices and encompasses organizational culture, national culture, language/politics/individual motivations and work ethics.

As mentioned above, the global distances make coordination, control and communication more challenging in globally distributed projects [39]. These three processes are defined as follows:
Table 1.1: Sourcing main characteristics.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location</strong></td>
<td>A sourcing can be delegated to a site in the same country, i.e. <strong>onshore</strong>, or to a site in another country, i.e. <strong>offshore</strong>.</td>
</tr>
<tr>
<td><strong>Legal entity</strong></td>
<td>Independently from the location, a sourcing can be transferred to a different branch (site) of the company, i.e. <strong>insourcing</strong>, or subcontracted to a different legal entity (company), i.e. <strong>outsourcing</strong>.</td>
</tr>
<tr>
<td><strong>Geographical distance</strong></td>
<td>In onshore projects, the geographical distance is considered: <strong>close</strong> when it is possible to have relatively frequent face-to-face meetings, since no flights are required to go from one site to the other; <strong>distant</strong> when at least one flight is required to have face-to-face meetings, which yields time and cost increases. In offshore projects, the geographical distance is considered: <strong>near</strong> when the required flying time is less than two hours; <strong>far</strong> when the flying time is longer than two hours and staying overnight is usually required.</td>
</tr>
<tr>
<td><strong>Temporal distance</strong></td>
<td>In onshore projects, the temporal distance is considered: <strong>similar</strong> when there is a time difference of one hour or less; <strong>different</strong> when the time difference between two sites is longer than one hour. In offshore projects, the temporal distance is considered: <strong>small</strong> when there is a time distance between sites of four hours or less; <strong>large</strong> when there is a time distance between two sites of more than four hours.</td>
</tr>
</tbody>
</table>
• **Coordination** is “the act of integrating each task with each organizational unit, so the unit contributes to the overall objective” [39].

• **Control** is “the process of adhering to goals, policies, standards or quality levels” [39].

• **Communication** is “a mediating factor affecting both coordination and control. It is the exchange of complete and unambiguous information” [39].

The global distances impact coordination, control, and communication processes in many different ways, as displayed in Table 1.2 [3].

### 4.2 Onboarding

In global long-living product development endeavors (the context of this thesis), the onboarding of software developers occurs many times during the endeavors’ life cycles. Given the challenges that are faced by developers in globally distributed projects, onboarding developers in this type of project is also very challenging [71, 72, 212].

Onboarding refers to the mechanism through which newcomers acquire the required knowledge, skills, and behaviors to become effective employees [10, 203]. Klein et al. [120] affirm that the research on onboarding can be divided into four distinct perspectives:

- Stages through which newcomers progress [33, 73].
- Actors involved with the onboarding of newcomers [6, 157].
- Tactics and practices employed by organizations for onboarding newcomers [9, 117, 203].
- Content to be learned by newcomers during the onboarding [42, 73].

Considering that the main focus of this paper is on onboarding tactics and practices, we elaborate further on this perspective, describing the main models of onboarding: Van Maanen and Shein’s model [203], Jones’ model [108] and Bauer’s model [9].

*Van Maanen and Shein’s Model*

Van Maanen and Shein [203] proposed a theoretical explanation regarding role orientation in the context of onboarding. The model categorizes onboarding tactics in six dimensions:

- **Collective vs. individual** – Collective onboarding occurs when a group of newcomers go through onboarding activities and acquire experiences together (e.g., boot camps). Individual onboarding occurs when newcomers go through separate from other newcomers (e.g., apprenticeship).
Table 1.2: Global distances impact on organizational unit processes (adapted from Ågerfalk *et al.* [3])

<table>
<thead>
<tr>
<th>Process</th>
<th>Temporal Distance</th>
<th>Geographical distance</th>
<th>Socio-cultural distance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coordination</strong></td>
<td>Coordination costs typically increase with distance.</td>
<td>Reduced informal contact can lead to reduced trust and a lack of critical task awareness.</td>
<td>Inconsistency in work practices can have a negative effect on coordination, as can reduce cooperation through misunderstandings.</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td>Management of project artifacts may be subject to delays.</td>
<td>Difficult to convey vision and strategy.</td>
<td>Perceived threat from training low-cost “rivals”. Different perceptions of authority/hierarchy can undermine morale. Managers must adapt to local regulations.</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>Reduced opportunities for synchronous communication, introducing delayed feedback.</td>
<td>Increased cost and logistics of holding face-to-face meetings</td>
<td>Potential for stimulating innovation and sharing best practice, but also for misunderstandings.</td>
</tr>
</tbody>
</table>
• **Formal vs. informal** – Formal onboarding relates to tactics in which newcomers are segregated from other employees (e.g., policy academies). Informal onboarding relates to tactics that have no or little separation between newcomers and other employees (e.g., on-the-job training).

• **Sequential vs. random** – Sequential onboarding refers to the extent to which discrete steps regarding the onboarding phases are specified for the newcomers, while random onboarding tactics do not specify any sequence of steps.

• **Fixed vs. variable** – Fixed onboarding occurs when there is a timetable associated with each step of the onboarding process, so that a newcomer knows the exact time required to complete each step. Variable onboarding does not associate any time with the onboarding steps. Rather, newcomers receive some clues regarding when they should consider an onboarding step as concluded.

• **Serial vs. disjunctive** – Serial onboarding takes place when experienced employees serve as models for newcomers (e.g., a new police officer works for an extended period with some veteran police officer). Disjunctive onboarding refers to the tactics wherein no guidelines or models are provided to newcomers.

• **Investiture vs. divestiture** – Investiture onboarding occurs when an organization prefers that newcomers keep their personal characteristics and make use of their own skills, values, and attitudes. Divestiture takes place when an organization rejects and removes the personal characteristics of newcomers.

According to this model, the way newcomers respond to their roles differs due to the onboarding tactics used by organizations. This means that organizations can support newcomers by giving relevant information in different ways.

*Jones’ Model*

Jones’ model [108] was built upon Van Maanen and Shein’s Model [203] and reduces the original six dimensions to two:

• **Institutionalized** onboarding occurs when tactics are implemented in structured programs and newcomers receive formal group orientation and mentoring. This dimension is composed by the following dimension categories of Van Maanen and Shein’s Model: collective, formal, sequential, fixed, and serial investiture.

• **Individualized** onboarding takes place when newcomers start working from the beginning and must learn the norms, values, and expectations on-the-fly. This dimension is composed of the following dimension categories of Van Maanen and Shein’s Model: individual, informal, random, variable, disjunctive, and divestiture.
Institutionalized onboarding is related to formal tactics, while individualized onboarding is related to informal tactics. Companies considered as successful regarding the onboarding of newcomers have more formal onboarding programs (institutionalized onboarding) [13, 36, 117].

Bauer’s Model

Bauer et al. conducted a series of studies that resulted in an empirically based onboarding model [9–13]. The model was conceptualized to support the design of onboarding programs, capitalizing on the fact that institutionalized onboarding is more successful than individualized onboarding [13, 36, 117].

While related to Van Maanen and Shein’s model and Jones’ model, Bauer’s model has a finer grain level than the previous models; it aggregates practices, techniques, methods and technologies (functions) that are related to successful onboarding (Figure 1.2). The benefits of this model are as follows: i) it facilitates the evaluation of current state of onboarding programs in real projects, supporting the identification of improvement areas; and ii) it provides a set of good practices that can be used by organizations to improve their onboarding programs. Thus, we considered Bauer’s model the most adequate to support answering the research questions of this thesis.

According to Bauer, onboarding has four distinct levels, known as the Four Cs, which are the building blocks of successful onboarding [9]:

- **Compliance** is related to teaching employees basic legal and policy-related rules and regulations.
- **Clarification** is related to ensuring that newcomers understand their new jobs and the related expectations.
- **Culture** is related to providing newcomers with a sense of organizational norms, including both formal and informal.
- **Connection** is related to the interpersonal relationships and information networks that newcomers must establish.

The extent to which an organization focuses on each C determines its onboarding strategy. The combination of tools, practices, recommendations, performance goals and measurement milestones constitutes an onboarding strategy, which is often formalized in an onboarding plan [9]. The success of an onboarding strategy is related to short-term and long-term outcomes. Short-term outcomes are associated with the adjustment of new employees to their new jobs [9]. They go through a series of four adjustments:

- **Self-efficacy** is the first level of adjustment and represents the degree that new employees feel confident when carrying out the work related to their new jobs. The more self-efficacy, the more motivated
and successful an employee has the potential to be [182]. Furthermore, self-efficacy is associated with high job satisfaction and low turnover [13].

- **Role clarity** is the second level of adjustment and represents how well new employees understand their role and expectations. Measures of role clarity are recognized as effective predictors of job satisfaction, organizational commitment and performance [74].

- **Social integration** is the third level of adjustment and represents the extent to which new employees feel socially comfortable and accepted by their colleagues and superiors [157]. Effective social integration is related to committed employees and low turnover rates [9].

- **Knowledge of culture** is the fourth level of adjustment and represents the possession of knowledge about the prevalent organizational culture (politics, goals, values and a company’s unique language) and the extent to which the new employees fit it [9].

Long-term outcomes of onboarding are related to attitudes and behaviors. Long-term successful onboarding is related to: higher job satisfaction, organizational commitment, low staff turnover, high performance levels, career effectiveness and low stress levels [9, 138, 146].

Companies that have been successful at onboarding newcomers employ a common set of practices, techniques, methods, and technologies (functions). These functions have been aggregated in a model, which was developed to support the development of onboarding strategies [9]. In the model (see Figure 1.2), the functions are categorized as follows:

- **Recruiting** – In many organizations, recruiting is not integrated with the onboarding plans and is treated as a separate function. However, existing literature [9, 10] shows that integration (e.g., through realistic job previews or early involvement of stakeholders) gives to candidates more accurate information about the company and the job. As a result, functions of this category facilitate the adjustment of new employees, especially self-efficacy, role clarity and knowledge of culture [119].

- **Orientation** – Formal orientation programs help newcomers to understand important aspects of their jobs and organizations, as the company’s culture and values [118]. Moreover, they also help newcomers feel welcome by presenting them to other individuals within the organization. Computer-based orientation programs can help to keep consistency to the program in different locations. Functions of this category facilitate all four types of adjustment (self-efficacy, role clarity, social integration, and knowledge of culture) [9].

- **Support tools and processes** – Tools and formal processes are of great value for onboarding success. According to Bauer [9], there
are three tools/processes that are related to successful onboarding: a written onboarding plan, which is a formal document that contains the timeline, goals, responsibilities, and support available to each newcomer; stakeholder meetings, which occur at specific intervals, involve all the onboarding stakeholders, and allow newcomers to get the information they need; and onboarding online, which can help to track the onboarding progress against development and career plans, and also help stakeholders to identify any additional help that new employees may need. Functions of this category facilitate all four types of adjustment factors (self-efficacy, role clarity, social integration, and knowledge of culture) [9].

- **Coaching and support** – Mentors can teach newcomers about the company, provide advice, and help with job instruction. Existing research shows that new employees with mentors acquire more knowledge about their new company than the ones without mentors [165]. Furthermore, mentoring programs and opportunities for informal interaction with colleagues help the new employees to adapt more easily to the new work environment. Functions of this category facilitate all four types of adjustment factors [9].

- **Training** – To give the newcomers the confidence, clarity, and skills required for their job, training is very important. Newcomers can receive training in hard skills and soft skills. The type of training depends on the self-efficacy of new employees in relation to what is demanded by the job. As a result, functions of this category facilitate the adjustment of new employees, especially self-efficacy, role clarity and knowledge of culture [9].

- **Feedback** – Newcomers need regular feedback and guidance to understand and interpret the reactions of their co-workers. Feedback can be mainly provided in two ways [9]: performance appraisals and 360-degree feedback, wherein the new employees are evaluated and receive developmental feedback; and employee-initiated information and feedback seeking, wherein the new employees proactively seek feedback. Functions of this category facilitate the adjustment of new employees, especially self-efficacy, role clarity and knowledge of culture [9].

4.3 Learning in the context of the onboarding software developers and teams

The onboarding of software developers throughout the life cycle of globally distributed projects leads to high demands for continuous learning.

Learning is defined as “the acquisition of knowledge or skills through study, experience, or being taught” [166]. Thus, learning is key for successful onboarding [9, 136].
According to Winterton et al. [219], there are three different types of learning: formal, non-formal and informal. **Formal learning** occurs within an organized and structured context. **Non-formal learning** occurs within planned activities that are not explicitly related to learning. **Informal learning** occurs by carrying out daily life activities.

Learning can also occur individually or in groups. **Individual learning** relates to the process through which a person acquires knowledge and skills individually. **Group learning** is a process of reflection and action, which encompasses different learning behaviors, such as asking questions, seeking feedback, experimenting, reflecting on results, and discussing errors or unexpected outcomes of actions [64]. It is often the case that team members learn together by testing assumptions and openly discussing different options to address existing problems [64].

Learning leads to the acquisition of knowledge, skills, and competences. **Knowledge** is the result of an interaction between the capacity and the opportunity to learn [219]. It is in general associated with formal learning, although can eventually be the result of non-formal or informal learning [219]. **Skill** is the combination of mental and physical capabilities that demand practice to acquire [219]. In many cases, obtaining knowledge is a pre-requisite for achieving a particular skill. It is in general associated with informal learning, although can eventually be the result of non-formal or formal learning [219]. **Competence** is the extent to which individuals interact effectively with the environment [219]. It describes personality aspects associated with better performance and higher motivation of individuals. It is acquired through informal learning [219].
Existing research shows that learning (formal, non-formal, and informal) is related to performance improvements \cite{60}. A lot of research is dedicated to show the relationship between informal learning and performance \cite{18, 99, 100, 114, 160, 199, 226}, also known as the learning curve phenomenon \cite{4}.

A learning curve describes the performance of teams or individuals in a mathematical way. It was proposed by Wright \cite{223} based on observations of how the costs associated with assembling airplanes decreased as the involved workers accumulated experience in doing the same type of task.

Learning curves can be modeled using univariate or multivariate models, e.g., log-linear, hyperbolic and exponential models \cite{4}. Log-linear models are most frequently employed due to their simplicity. The original model proposed by Wright \cite{223} is a log-linear model, which is represented by Equation 1.1.

\begin{equation}
Y = CX^b
\end{equation}

where $Y$ is the average time (or cost) per unit demanded to produce $X$ units (cumulative experience) and $C$ is the time (or cost) to produce the first unit. The parameter $b$, also known as learning rate, represents the slope of the learning curve. Many researchers have proposed modifications to Equation 1.1 (e.g., the De Jong and S-curve models \cite{4}), in general focusing on how the cumulative experience is computed or exploiting the learning rate.

5 RELATED WORK

Studies with some relationship with the onboarding of software developers and teams in globally distributed projects often focus on comparing offshore teams with the original experienced developers in the prime development location. Time to acquire the required knowledge to perform as expected is of particular interest for companies that make decisions to onboard developers and teams in offshore locations. Mockus and Weiss \cite{149} studied individual developers working on non-trivial modification requests and found that offshore developers may reach full productivity in approximately 15 months, after three months of project training prior to the actual work. Perception-based studies and experience reports suggest that the learning process may take from 12 months \cite{62}, up to three \cite{215} and five years \cite{122, 213} or even longer \cite{17}, leading to longer onboarding than in collocated projects.

Many studies have reported empirical quantitative investigations about the relationship between informal learning and performance \cite{18, 99, 100, 114, 160, 199, 226}. Huntley \cite{100} explored the relationship between infor-
mal learning and debugging cycle times in open source projects. The results suggest that the accumulation of experience impacts debugging cycle times more in mature projects than in emerging projects. Tüzün and Tekinerdogan [199] investigated the impact of informal learning on ROI (return on investment) in software product line engineering. They concluded that the learning curve has a clear impact on the ROI of software development companies, although such an impact gets lower when the number of products of a particular software product line increases. Zorgios et al. [226] proposed an explanatory theory for team learning related to software development. The authors modeled the interaction between learning rates of development teams and improvements in their productivity.

Other studies focus on the role of mentoring, technical factors and social relationships when onboarding software developers in globally distributed open source projects. Fagerholm et al. [71, 72] investigated the impact of mentoring on the onboarding of new developers. It was found that developers who received deliberate onboarding support through mentoring were more active in the beginning. Labuschagne and Holmes [128] investigated the effectiveness of onboarding strategies employed in open source projects supported by the Mozilla Foundation. They measured effectiveness as the number of long-term contributors. The strategies were implemented as two different programs: one with focus on easy bug fixing without mentoring, and the other focused on mentored bug fixing of more complex bugs. It was found that only mentoring was not enough to increase the number of long-term contributors.

A few studies focus on the barriers faced by newcomers and appropriate approaches to start contributing to open source projects. Steinmacher et al. [191, 193] investigated the barriers faced by newcomers to contribute in globally distributed open source projects and the challenges associated with selecting the initial task to start contributing. They identified 15 barriers that hinder the onboarding process of new developers in open source projects, which were classified into five categories (social interaction, newcomers’ previous knowledge, finding a way to start, documentation, and technical hurdles) and have three origins (newcomers, community, and product). Furthermore, they found that new developers are not confident to choose their initial task and, thus, need support from the open source community to select an appropriate task.

In summary, existing literature shows that:

- The onboarding of software developers in globally distributed projects may take longer time as compared to onboarding in collocated projects.
- Learning is a key component of onboarding and is essential for newcomers to achieve good levels of performance.

4 www.mozilla.org/en-US/foundation/
• Mentoring is related to better onboarding results, although it needs to be used together with other onboarding functions.
• Newcomers have to face many barriers before they are able to start working.

6 RESEARCH GAPS

Although existing literature suggests that onboarding of developers in globally distributed projects is more challenging than in other contexts (as detailed above), most of the studies either are based on perceptions/-expert opinion or were conducted in contexts such as open source projects or small-scale projects. Furthermore, no study has approached onboarding in a holistic way, accounting for all onboarding function categories at the same time. Finally, there are no approaches to support the development of strategies to onboard developers/teams and evaluate onboarding results in a systematic way, which is found as very important to be successful when carrying out the onboarding of other roles (e.g. managers) [9].

This thesis fills the existing gaps by employing quantitative and qualitative analysis to identify the challenges and implications of onboarding developers and teams in the target context (large-scale globally distributed legacy projects). Furthermore, we propose a measurement approach focused on productivity, autonomy, and maturity to support the development of onboarding strategies and evaluation of onboarding results. Finally, recommendations were proposed, practices, tools, techniques, methods, and technologies were identified, and a process was proposed to support the development onboarding strategies and evaluation of onboarding results.

7 RESEARCH DESIGN AND METHODS

The research problem and associated research questions of this thesis were addressed through an empirical research approach. The conduction of the investigations in real world cases means that the research reported in this thesis has a high level of relevance [103], although the level of rigor is not as high due to the limitations of the employed research methods (mainly case study) [103]. Thus, rather than focus on proposing new theories, this thesis focused on widening the understanding about the challenges and implications associated with the defined research problem. Furthermore, it also focused on providing empirically based solutions to address the outlined research problem.

The papers included in this thesis have distinct objectives, research questions, and analysis perspectives. Therefore, the individual contributions of each paper were combined to address the research questions of this the-
sis in a process to support the development of onboarding strategies and evaluation of onboarding results (see Section 9). The perspective associated with the research problem and research questions of this thesis enabled the interpretation and combination of the papers in a more holistic manner. This helped to provide a higher order understanding of the challenges and implications related to the onboarding of software developers and teams in globally distributed legacy projects. Finally, this also allowed to combine the solutions proposed in each paper, aiming at addressing the defined research problem, which was operationalized through using business process modeling (BPM) [96].

To address the research questions of this thesis, we employed a mixed method approach. The following research methods were employed: literature review and case study. Furthermore, we used BPM [96] to propose the process presented in Section 9. The remainder of this section presents more details about the used methods and the cases we investigated in this thesis.

7.1 Case study

Case study is a research method that can be operationalized using different approaches, such as Yin’s approach [224], Merriam’s approach [145], and Stake’s approach [189]. In this thesis, we employed Yin’s approach, since it is most used case study approach by the software engineering research community [181].

Note that methods such as experiment [221] or survey [76] were not considered due to the following reasons:

- Experiments require controlled environments. If on one hand it allows for high level of rigor [103], it does not allow to investigate a given phenomenon in its natural context.
- Surveys allow for the investigation of large populations [76], but do not enable in-depth investigation of a phenomenon in its natural context.

We selected case study as the main research method of this thesis because it is recommended when it is necessary to investigate individual, group, organizational, and social phenomena [224]. The case study method supports the investigation of a contemporary phenomenon within its natural context [181]. While case study enables a high level of realism, it sacrifices the level of rigor [103].

Case studies are often conducted through the following phases [181, 224]:

- **Design**, wherein the case study is planned.
• **Preparation** for data collection, wherein data collection procedures are established.
• **Collecting** evidence, wherein the data collection is performed.
• **Analysis** of collected data, wherein the collected data is analyzed using appropriate methods.
• **Reporting**, wherein the findings are reported.

Both qualitative and quantitative data can be used in case studies. To increase the credibility level associated with case studies, researchers employ triangulation of data sources, methods, observers, and theories [181].

In case studies, data is often collected through interviews, workshops, observations and archival research [181]. In this thesis, we mainly used semi-structured and unstructured interviews, workshops and archival research.

According to Yin, there are four types of case study. A case study can be **single** or **multiple**. Furthermore, multiple or single case studies can be either **holistic**, when there is just a single unit of analysis, or **embedded**, when there are multiple units of analysis. Finally, case studies can be **exploratory**, when one tries to seek new insights about a given phenomenon, **explanatory**, when one tries to find explanations for a given phenomenon, or **improving**, when one tries to improve something of a given phenomenon.

In this thesis, we conducted one exploratory holistic single-case study, which is reported in P2–P4 and P6, and one exploratory holistic multi-case study, which is reported in P5. In the first case study, we investigated one case from different perspectives (measurement approach, mentoring, learning, and effort estimation), while in P5 we investigated three different cases from the same perspective (a holistic view of the investigated onboarding strategies). In both cases, the context was the same: large-scale globally distributed legacy projects.

Chapters 3–7 provide more details about the design of each study.

### 7.2 Cases in this thesis

We investigated three companies in this thesis (Ericsson, one Norwegian company and one Polish company5). One case from Ericsson was investigated in one holistic case study from multiple perspectives (P2–P4 and P6). Furthermore, we investigated the same case from Ericsson in combination with the cases from two other companies in one multi-case study.

We selected the cases through convenience sampling in consultation with representatives from each company as cases suitable for studying the challenges and implications associated with the onboarding of developers

---

5 Due to non-disclosure agreements, the two companies were anonymized in this thesis and in the associated research paper.
and teams in the target context. We describe the investigated cases in the remainder of this section\(^6\).

**The Ericsson case**

This case is a large-scale distributed endeavor associated with the development and maintenance of a large telecommunication software product in Ericsson. Its degree of global distribution increased significantly during the period covered in our investigation. The product originated in Sweden, has evolved for almost 20 years, and comprises a considerable amount of complex legacy code. It was subject to many technical and methodological changes over the years, like introducing a new programming language (Java in addition to C++), introducing a new testing technology (TTCN-3\(^7\)) and changing the software development methodology (from plan-driven to agile).

For our investigation, we covered the teams that were still active in October 2016. In total, this involved 188 employees, including 15 software architects and 134 developers working in 19 formal teams. The teams were located in Sweden (five teams), India (10 teams), Italy (one team), USA (one team) and Poland (two teams). Figure 1.3 shows the project evolution in terms of sites. The numbers in the circles represent the number of teams at each site. Offshore locations were added to address the growing demands for resources and to implement market-specific customizations. The teams in India (10 teams) and Poland (one team) were onboarded relatively recently. The sites in China and Turkey (one team each) were closed down due to business reasons and are not included in the analysis. The company transferred the main responsibility for the product to India in 2017. Note that 12 developers were onboarded in India during spring 2016 (seven in March 2016 and five in May 2016) to replace staff attrition.

The software development teams are cross-functional and use agile practices in their daily work. The teams have four to seven developers and a design lead, who is a senior developer. Due to the scale and level of distribution of the case, the involved project managers use a mix of agile and plan-driven practices to manage and coordinate the work of the software development teams.

The software development teams receive tasks, which can be product customizations, maintenance, product improvements, standardizations of market features and business use cases. Each task resembles an independent project with a specific start and end date and expected results. One or more development teams are assigned to each task. For example, the sched-

\(^6\) Note that more details were provided regarding the Ericsson case since it we were allowed to provide such a level of detail.

\(^7\) [www.ttcn-3.org](http://www.ttcn-3.org)
The evolution of a product customization task (PC) varies from one to six months, while business use cases (BUC) may take from three to twelve months. Scale and distribution of the selected case also influence how the product software architecture is managed and how product knowledge is transferred to developers [27]. Architecture management is centralized in Sweden, where a team of software architects ensures the integrity and evolvability of the product architecture. Moreover, the team of architects supports all development teams by responding to architectural questions and by providing feedback on the teams’ work through code reviews. In urgent or particularly complex situations, the software architects may also participate in actual code implementation.

**The Norwegian case**

This case is related to a leading supplier of intelligent transportation systems to the public transport sector, including fare collection, travel information, infotainment, fleet management and traffic management. The company has subsidiaries in Poland, Sweden, and Norway. The software development teams follow a Scrum-based process and have a complex portfolio of products. Most products have a large amount of legacy code.
The Polish case

This case is related to a Polish company whose main market segments are telecommunications, M2M (Machine to Machine), healthcare ERP (Enterprise Resource Planning), business intelligence applications, and finances and banking solutions. The company has subsidiaries in Sweden, Ukraine, and Belarus. Our investigation focused on the people involved in the development of telecommunication solutions. The software development teams follow a Scrum-based process and have to deal with legacy code very often.

7.3 Business process modeling

A way to combine the different findings from the studies included in this thesis is by packaging them as a process. To develop the process, we used BPM [96]. The resulting process provides a roadmap on how use the proposed solutions to strategize onboarding undertakings and evaluate onboarding results (see Section 9 for more details).

BPM is the activity of representing the business processes of an organization [209]. A business process is defined as the combination of related activities to achieve a specific goal (e.g., product or service). A business process can be used to improve or facilitate the learning of existing processes. According to von Rosing et al., there are three main types of business processes [209]:

- **Management processes** – Processes such as corporate governance, and strategic management, which govern the operation of an organization.
- **Operational processes** – Processes like purchasing, manufacturing, marketing, and sales, which are the core business of an organization and create the primary value stream.
- **Supporting processes** – Processes such as accounting, recruitment, and onboarding, which support the core business of an organization. This is the type of process proposed in this thesis.

To carry out the process modeling, we used the knowledge accumulated during the conduction of the investigations reported in this thesis. Thus, we conducted the following steps (Figure 1.4):

1. We selected from the individual papers the solutions that could be used as activities into the process to support the development of onboarding strategies and evaluation of onboarding results.
2. We identified additional activities or artifacts required to enable a coherent process flow.
3. We combined all activities, defining the appropriate sequence for their execution and resulting artifacts.
4. We received feedback about the process provided by a domain expert (an R & D manager).

The solutions from the individual papers were combined accounting for the following:

- Before strategizing the onboarding of new developers and/or teams, it is important to identify the context wherein the undertaking will take place [168]. In doing so, decision-makers can identify particularities of a context early on (e.g., cultural aspects) and account for them when developing onboarding strategies. Furthermore, by describing the context, it becomes easier to mine knowledge [204] acquired through previous onboarding undertakings.

- It is important to track the progress of an onboarding undertaking, to evaluate whether the performance of onboarded developers/teams (onboarding results) is evolving as expected. To do so, it is necessary to define how to measure performance and performance goals.

- To avoid “reinventing the wheel” or repeating the same mistakes, it is important to document and reuse acquired knowledge related to previous onboarding undertakings.

We combined the solutions from P1–P5 and added additional activities and artifacts to enable knowledge reuse. Finally, the resulting process was presented to an R & D manager from Ericsson, who is actively involved with the onboarding of software developers and teams. The process was considered as useful by the manager. The resulting process is presented in Figure 1.5 and detailed in Section 9.

7.4 Literature review

The literature review method supports the identification and aggregation of knowledge from existing literature. It can be conducted in a systematic way, which is known as systematic literature review (SLR) [115]. An SLR encompasses three main phases [115]:

- In the Planning phase, the review protocol is designed. The review protocol contains the research questions, the search strategy to collect relevant studies, the criteria to select primary studies and the criteria to evaluate the quality of selected primary studies.

- In the Conducting phase, related studies are identified, selected, evaluated and synthesized based on the review protocol.

- In the Reporting phase of the SLR method consists on communicating the achieved results by means of a written document.

In P1 (Chapter 2), it was required to identify relevant dimensions to classify and document globally distributed projects, and, thus, classify, de-
Figure 1.4: Process modeling.
scribe and document contexts related to onboarding undertakings in globally distributed projects. To identify such dimensions, we conducted an SLR in a previous study [24]. The conducted SLR was later on complemented with a regular literature review. More details about the research design of P1 are presented in Chapter 2.

8 CONTRIBUTIONS AND SUMMARY OF INCLUDED RESEARCH PAPERS

The individual research papers included in this thesis have their own contributions, however, this thesis has the following general novel contributions:

1. A holistic view on the challenges and implications related to the onboarding of software developers and teams in large-scale globally distributed legacy projects (RQ1). This contribution resulted from the combination of the findings of P2–P6.
2. A process to support the development of onboarding strategies and evaluation of onboarding results (RQ2). This contribution resulted from the combination of the solutions put forward in P1–P5.

In the remainder of this section, we present a summary of the included research papers and their respective main contributions.

8.1 Chapter 2 (P1) – An extended global software engineering taxonomy

In global software engineering (GSE), the need for a common terminology and knowledge classification has been identified to: i) facilitate the sharing and combination of knowledge by GSE researchers and practitioners; and ii) to allow for the classification and documentation of projects’ contexts. A GSE taxonomy was recently proposed to address such a need, focusing on a core set of dimensions. However, its dimensions did not represent an exhaustive list of relevant GSE factors.

The main objective of P1 is to extend the existing taxonomy, incorporating new GSE dimensions that were identified by means of two empirical studies. To address the research questions of the study, we used evidence found through a systematic literature review and a survey and added new dimensions to the existing taxonomy.

We identified seven dimensions to extend and incorporate into the original GSE taxonomy. The resulting extended taxonomy was later on validated by: i) comparing it with the original taxonomy. By doing so, we highlighted how it complemented the original taxonomy; and ii) demonstrating its utility using it to classify eight finished real GSE projects. The extended taxonomy was representative enough to classify the projects in a clear way.
The main contribution of this study is:

1. An extended taxonomy developed to enable the classification, documentation and knowledge mining of globally distributed projects.

8.2 Chapter 3 (P2) – Experiences from measuring learning and performance in large-scale distributed software development

Developers and teams in large-scale distributed software development are often required to learn continuously. Organizations also face the need to train and support new developers and teams onboarded in ongoing projects. Although learning is associated with performance improvements, experience shows that training and learning do not always result in a better performance or that significant improvements might take too long.

In P2, we reported experiences from establishing an approach to measure learning results and associated performance impact for developers and teams in Ericsson. While deploying a measurement approach, we faced many challenges, which were documented in the form of lessons learned.

Measuring learning and performance is a challenging task. The main identified challenges were related to data inconsistencies caused by, among other factors, the distributed nature of the project.

The main contributions of this study are:

1. The identification of challenges and implications associated with deploying a measurement approach to evaluate onboarding results through performance.
2. A measurement approach for evaluating onboarding results through team productivity.

8.3 Chapter 4 (P3) – Software architects in large-scale distributed projects: An Ericsson case

Software architects are recognized as key assets for successful software development projects. However, not much has been published about the challenges associated with the work of software architects in large-scale distributed projects, especially regarding mentoring newly onboarded developers/teams and guarding architectural integrity.

The main objective of P3 was to shed light on this topic by: i) identifying how software architects are organized; ii) their roles and responsibilities; and iii) the amount of effort they spend on mentoring newly onboarded developers/teams and guarding a large-scale legacy product developed
by teams from multiple distributed locations in Ericsson. To address the research questions of this study, we conducted a case study in Ericsson.

The results show that to deal with the challenges of scale, distribution and monolithic architecture of a legacy software product, it is required to follow a more centralized approach, i.e. the architectural decisions are centralized to a team of architects. Code reviews are extensively used not only to check the state of the code, but also to reveal defects that have potential to turn into maintainability problems. Our results also suggest that the effort spent by architects on designing, guarding architecture integrity and evolvability, and mentoring software development teams is directly related to team maturity. This means that significant investments are needed whenever new teams and locations are being onboarded.

The main contributions of this study are:

1. The identification of challenges and implications related to mentoring immature software developers and teams onboarded in offshore locations.
2. A measurement approach for evaluating onboarding results through team maturity.

8.4 Chapter 5 (P4) – Learning and performance evolution of immature remote teams in large-scale software projects: An industrial case study

Large-scale distributed software projects with long life cycles often involve a considerable amount of complex legacy code. The combination of scale and distribution challenges, and the difficulty in acquiring knowledge about large amounts of complex legacy code may make the onboarding of new developers/teams problematic. This may lead to extended periods of low performance of these developers/team.

The main objective of P4 is to analyze the learning process and performance evolution (team productivity and team autonomy) of remote software development teams added late to a large-scale legacy software product development, and to propose recommendations to support the learning of remote teams. To address the research questions of this study, we conducted a case study in Ericsson.

Our results show that the productivity and autonomy of immature remote teams are on average 3.67 and 2.27 times lower than the ones of mature teams, respectively. We also identified four challenges that affected the learning process and performance evolution of immature remote teams: complexity of the product and technology stack, distance to the main source of product knowledge, lack of team stability, and training expectation misalignment. Our results indicate that scale, distribution and complex legacy code may make learning more difficult and demand a long
period to achieve high performance. To support the learning of remote teams, we put forward five recommendations.

The main contributions of this study are:

1. The identification of challenges and implications associated with the learning process and performance evolution of immature software developers and teams onboarded in offshore locations.
2. A measurement approach for evaluating onboarding results through team autonomy.
3. Recommendations to support the learning process of software developers and teams in large-scale globally distributed legacy projects.

8.5 Chapter 6 (P5) – Onboarding software developers and teams in three globally distributed legacy projects: A multi-case study

Onboarding is the process of supporting new employees regarding their social and performance adjustment to their new job. Software companies have faced challenges with recruitment and onboarding of new team members. Effective onboarding helps newcomers to acquire attitudes, knowledge, skills, and behaviors required to work effectively. While onboarding in a diverse type of roles/professions has been investigated in other research fields, there is no study that investigates in a holistic way how the onboarding of software developers is strategized by software companies.

The main goal P5 is to investigate the strategies of onboarding software developers and teams in globally distributed projects. We conducted a multi-case study, which involved three companies (a Swedish site of Ericsson, a Polish company, and a Norwegian company) and employed Bauer’s model for successful onboarding to identify the current state of the onboarding strategies employed in each company.

The results show that the employed strategies do not cover the whole repertoire of Bauer’s model (see Section 4), while the ones that are covered are semi-formalized. We conclude that there is room for improvement in the investigated companies. Besides, in projects with multiple sites, some functions are executed locally and the onboarding outcomes may be hard to control. Our results also suggest that the onboarding of new developers/teams in remote locations of distributed legacy projects is even more challenging than onboarding locally. This may be further amplified by project scale. Finally, we identified practices that can support the onboarding of software developers and teams in globally distributed projects.

The main contributions of this study are:
1. The identification of challenges and implications related to developing onboarding strategies for projects that involve multiple sites and legacy code.

2. Recommendations and a set of practices, tools, techniques, methods, and technologies to support the development of successful onboarding strategies.

8.6 Chapter 7 (P6) – Effort estimation in large-scale distributed agile software development: An industrial case study

Frequently, schedule and budget overruns occur in software projects. Planning and estimation are particularly challenging in large and globally distributed projects. While software engineering researchers have been investigating effort estimation for many years to help practitioners to improve their estimation processes, there is little research about effort estimation in large-scale distributed projects.

The main objective of P6 is three-fold: i) to identify how effort estimation is carried out in large-scale distributed projects; ii) to analyze the accuracy of the effort estimates in large-scale distributed agile projects; and iii) to identify how factors such as team maturity, scale, and distribution relate to schedules overrun in large-scale distributed agile projects. To address the research questions of this study, we conducted a case study in Ericsson.

Our results show that immature teams incur larger schedule overruns than mature teams and do so more often. Furthermore, we found that requirements with large size/scope, low priority and developed in multi-site settings are related to larger schedule overruns and do so more often. Thus, immaturity of newly onboarded teams and the challenges associated with the scale, distribution and legacy code add complexity to effort estimation.

The main contribution of this study is:

1. The identification of challenges and implications associated with estimating software development effort in large-scale globally distributed legacy projects that involve immature developers and teams onboarded in offshore locations.

9 The proposed process

The resulting process is presented in Figure 1.5. It has five activities and a knowledge repository, whose role is to allow for documentation and reuse of onboarding experiences and lessons learned. The knowledge repository can be implemented in different forms, like a wiki. This process was designed to be used within the context of globally distributed legacy projects,
although it can be adapted to be used in any other context wherein software developers and teams are onboarded.

The process works as a recommended way to use the solutions proposed in this thesis. However, it does not represent the only way to employ the proposed solutions; decision makers should consider which activities fit their respective contexts best. The activities and an example of how to use the process are presented in the remainder of this section.

9.1 Define context

Classifying, describing and documenting the context of an onboarding undertaking is essential to allow for identifying the associated needs, particularities and challenges [168]. It also allows for a more systematic reuse of previous onboarding experiences and facilitates knowledge mining [204].

Define context is the first activity of the proposed process. In this activity, an onboarding undertaking context is classified using the extended version [28] (P1, Chapter 2) of Smite et al.’s GSE taxonomy [214], whose dimensions are presented in Tables 1.3, 1.4 and 1.5. The taxonomy allows for classification at the site level (Table 1.3) and the relationship-between-pair-of-sites level (Tables 1.4 and 1.5). In the context of the proposed process, the taxonomy is to be used following these steps:

1. Identify the source site(s) (the main source(s) of knowledge and where the key decision makers are located) and the target sites (locations where new software developers/teams will be onboarded). Note that it can be the case that the target and source sites are the same, which happens when new teams will be onboarded in the same location where the main source of knowledge and the decision makers are located.
2. Classify each involved site using the dimensions described in Table 1.3. Note that the way we obtained the thresholds in Table 1.3 is detailed in Chapter 2.
3. Classify each relationship between a pair of sites using the dimensions described in Tables 1.4 and 1.5. Note that if the source and target sites are the same, this step is not necessary.

More details about the taxonomy are presented in Chapter 2.

9.2 Strategize onboarding undertaking

Before starting any onboarding undertaking, it is important to define the appropriate strategy for the associated context. The combination of tools, practices, recommendations, performance goals and measurement mile-

I
Figure 1.5: The proposed process.
Table 1.3: Dimensions of the GSE taxonomy at the site level.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language distance</td>
<td>No distance, Small, Medium, large</td>
<td>The language distance is measured in terms of the distance between a site’s mother tongue and English (language distance index), which is a number that varies from 0 to 1. There is <strong>no distance</strong> when a site’s mother tongue is English or there is no need for a lingua franca. The distance is <strong>small</strong> when the a site’s language distance index is smaller or equal to 0.4. It is considered <strong>medium</strong> when a site’s language distance index is greater than 0.4 and smaller than 0.57. Finally, it is considered <strong>large</strong> when it is greater than 0.57 and smaller or equal to 1.</td>
</tr>
<tr>
<td>Software process type</td>
<td>Agile, Plan-driven</td>
<td>A site is to be classified as <strong>agile</strong> if its software process is mainly based on agile practices. Otherwise, it is to be classified as <strong>plan-driven</strong>.</td>
</tr>
<tr>
<td>Power distance</td>
<td>Small, Large</td>
<td>A site has <strong>large</strong> power distance when its power distance index is greater than 50; otherwise, it is considered <strong>small</strong>.</td>
</tr>
<tr>
<td>Uncertainty avoidance</td>
<td>Weak, Strong</td>
<td>A site has <strong>strong</strong> uncertainty avoidance when its uncertainty avoidance index is greater than 63; otherwise, it is considered <strong>weak</strong>.</td>
</tr>
</tbody>
</table>
Table 1.4: Dimensions of the GSE taxonomy at the relationship-between-pair-of-sites level - Part I.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Onshore, Offshore</td>
<td>A sourcing can be delegated to a site in the same country, i.e. onshore, or to a site in another country, i.e. offshore.</td>
</tr>
<tr>
<td>Legal entity</td>
<td>Insourcing, Outsourcing</td>
<td>Independently from the location, a sourcing can be transferred to a different branch (site) of the company, i.e. insourcing, or subcontracted to a different legal entity (company), i.e. outsourcing.</td>
</tr>
<tr>
<td>Geographical</td>
<td>Close, Distant, Near, Far</td>
<td>In onshore projects, the geographical distance is considered: close when it is possible to have relatively frequent face-to-face meetings, since no flights are required to go from one site to the other; distant when at least one flight is required to have face-to-face meetings, which yields time and cost increases. In offshore projects, the geographical distance is considered: near when the required flying time is less than two hours; far when the flying time is longer than two hours and staying overnight is usually required.</td>
</tr>
<tr>
<td>Temporal distance</td>
<td>Similar, Different, Small, Large</td>
<td>In onshore projects, the temporal distance is considered: similar when there is a time difference of one hour or less; different when the time difference between two sites is longer than one hour. In offshore projects, the temporal distance is considered: small when there is a time distance between sites of four hours or less; large when there is a time distance between two sites of more than four hours.</td>
</tr>
</tbody>
</table>
Table 1.5: Dimensions of the GSE taxonomy at the relationship-between-pair-of-sites level - Part II.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software process distance</td>
<td>Equal, Similar,</td>
<td>The software processes of two sites are considered <strong>equal</strong> when they use the same workflows, roles, and practices to develop software. They have <strong>similar</strong> processes when they have the same software process type, but the workflows, roles, and practices are not exactly the same. The processes are considered different when there are no commonalities.</td>
</tr>
<tr>
<td></td>
<td>Different</td>
<td></td>
</tr>
<tr>
<td>Communication model</td>
<td>Low synchronicity, high synchronicity, balanced synchronicity</td>
<td>It is <strong>low</strong> when the communication is mainly based on asynchronous media (e.g. email). It is <strong>high</strong> when the communication is mainly based on synchronous media (e.g. instant messaging tools). It is <strong>balanced</strong> when the communication model has both synchronous and asynchronous media and each media type is used for its most adequate purpose.</td>
</tr>
</tbody>
</table>
stones constitutes an onboarding strategy, which is formalized in an onboarding plan.

To strategize an onboarding undertaking, the decision makers must account for the tools, practices, methods and techniques identified in P5 [29] (Chapter 6) and the recommendations proposed in P4 [30] (Chapter 5), such as:

- Integrate recruitment and onboarding.
- Provide realistic job previews.
- Create an onboarding plan.
- Involve key stakeholders in the recruitment and onboarding process.
- Provide formal orientation for newcomers.
- Provide mentoring for newcomers.
- Evaluate the progress of newcomers.
- Provide feedback for newcomers.
- Take into account cultural differences when developing the learning process.
- Ensure collocated mentoring to support the learning process of offshore teams when there is not enough competence locally.
- Facilitate group learning.
- Use code reviews to support learning.

To define performance goals and milestones, it is necessary to define performance indicators of interest and baselines. As of P2 – P5, we defined productivity, autonomy and maturity as performance indicators (Chapters 3 – 6). In addition to the performance indicators, we also defined a metric called task size, which is a key metric in the proposed measurement approach. The performance indicators and task size are defined as follows:

- **Task size** is measured in complexity points and takes into account a task's extent and complexity to make tasks of different degrees of complexity easier to compare.
- **Productivity** represents the effort that a developer or team spends to complete a complexity point.
- **Autonomy** represents how independently a developer or team fulfills a task.
- **Maturity** represents developer/team maturity. Maturity is related to the amount of knowledge associated with a legacy product under development/maintenance, and knowledge about the associated technology stack. The bigger the knowledge, the bigger the maturity.

The performance goals for a developer or team are defined based on the average performance of high performance developers or teams (baseline), with the addition of some accepted margin of variation (up and down the
average, to be defined by the decision makers when strategizing an onboarding undertaking). Performance is measured using the measurement approach proposed in this thesis (see Chapters 3–5 for more details).

More details about the tools, practices, recommendations, and measurement approach are presented in Chapters 3–6.

9.3 Evaluate onboarding results

To identify whether an onboarding undertaking is progressing as expected, it is necessary to: i) measure the performance of onboarded developers or teams using the performance indicators described above; and ii) compare the result with the defined performance goals. If the onboarding undertaking is not progressing as expected, it may be the case that the company needs to act (e.g., providing more formal training). The metrics associated with each performance indicator are presented in the remainder of this section.

**Task size** $T_i$ is measured as the size of task $i$ and $T_{ki}$ is the size of the proportion of task $i$ that was completed by team $k$ or developer $k$.

**Effort** $E_{ki}$ is the total effort spent by team $k$ or developer $k$ on task $i$.

**Mentoring Effort** $E_{hki}$ is the effort in work hours spent by software architects providing mentoring and help to team $k$ or developer $k$ during the fulfillment of task $i$.

**Productivity** $P_{ki}$ is measured as the productivity of team $k$ or developer $k$ on task $i$:

$$P_{ki} = \frac{T_{ki}}{E_{ki}} \quad (1.2)$$

**Autonomy** $A_{ki}$ is measured as the autonomy with which team $k$ or developer $k$ carried out task $i$:

$$A_{ki} = \frac{T_{ki}}{E_{hki} + 1} \quad (1.3)$$

**Maturity** $M_{ki}$ is measured as the maturity of team $k$ or developer $k$ before carrying out task $i$. It is measured using the authority matrix presented in $P_3$ [27] (Chapter 4). The authority matrix is showed in Figure 1.6.

The matrix has a learning and mentoring curve that shows how the responsibilities of software architects and developers/teams change as they get more mature. In its y-axis, there are the four main activities that developers/teams and architects are involved, while the x-axis has the four defined maturity levels [27]:

- **Level A** – A developer or team on this level has no or very basic knowledge about the code and architecture of the product. They require a lot of mentoring and support from the architects, even when
implementing (i.e. coding, testing and documenting) non-complex technical solutions. Software architects guide the Level-A developers/teams during the implementation and are actively involved, both with reviewing and approving code.

- **Level B** – A developer or team on this level has sufficient knowledge to implement non-complex technical solutions without much guidance. More knowledgeable team members and design leads of Level-B teams are able to review the code implemented by the team, but software architects are still required to review a majority of the design and code.

- **Level C** – It represents experienced and mature developers or teams with good knowledge of the product architecture, which are able to implement complex technical solutions that have a significant architectural impact. They are capable of reviewing and approving their own code and software architects are only involved in approval when critical components of the software architecture are affected (a critical component contains core functionality of the product that executes key operations). More solution design work is also delegated to Level-C teams. Software architects support these Level-C teams/developers by mentoring the design of technical solutions and providing on-demand guidance in the initial stages of implementation.
• **Level D** – It represents very experienced and rather autonomous developers or teams that are capable of implementing complex technical solutions independently, even the ones that affect critical components of the product architecture. Code implemented by them does not need to be approved by software architects. They can also drive the design and review of technical solutions. Software architects are involved only in the technical solution design.

More details about the measurement approach are presented in Chapters 3 – 5.

### 9.4 Identify and address issues

The results of the previous activity (evaluate onboarding results) must be compared to the defined performance goals. Deviations can indicate that there are issues in the onboarding undertaking that need to be addressed. Issues may be related to insufficient amounts of formal training and insufficient amounts of mentoring. It may be the case that it is not worthwhile for a company to invest time and money to address an issue. For example, when a software development team is presenting poor performance after many measurement milestones, decision makers may decide to close down the associated site.

An onboarding undertaking is considered as successful when the onboarded developers or teams achieved the defined performance goals. The issues and associated solutions (if any) should be documented in a knowledge repository (e.g., wiki).

### 9.5 Using the process

To illustrate the usage of the process, we use as example the onboarding of five product customization teams in India in the context of the Ericsson case (see Section 7.1). Two of the five teams were onboarded in August 2014, one in January 2015 and two more in August 2015. The onboarding timeline for the five teams is summarized in Figure 1.7.

In this example, there are two sites (Figure 1.8), one located in Sweden (A – source site) and the other located in India (B – target site). Since there are just two sites in this example, there is just one relationship (AB). The result of employing the taxonomy to describe the context of the example is presented in Tables 1.6 (site level) and 1.7 (relationship-between-pair-of-sites level).

The onboarding was strategized in the following way:
Figure 1.7: Onboarding timeline for product customization teams in India [30].

Figure 1.8: Sites involved in the example.
• The average productivity, autonomy and maturity level of high-performance teams located in Sweden, Italy, and the USA were used to define the performance goals of the new teams.

• Training and mentoring were provided on site in Sweden for the first two Indian teams, so that these teams could help with the onboarding of other teams later on. This initial training in Sweden was followed by on-site mentoring provided by senior software developers. They went from Sweden to India and stayed for four months. The senior developers also provided training for a team onboarded during this period.

• The onboarding was mainly strategized by the decision makers located in Sweden, but the recruitment process was planned and conducted by local personnel in India.

• The performance of the onboarded teams was to be continuously evaluated, and they were to take the main responsibility for the product after two years.

Table 1.6: Classification at the site level for the example.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language distance</td>
<td>No distance (all employees had to be fully fluent in English)</td>
<td>No distance (all employees had to be fully fluent in English)</td>
</tr>
<tr>
<td>Software process type</td>
<td>Agile</td>
<td>Agile</td>
</tr>
<tr>
<td>Power distance</td>
<td>Small</td>
<td>Large</td>
</tr>
<tr>
<td>Uncertainty avoidance</td>
<td>Strong</td>
<td>Week</td>
</tr>
</tbody>
</table>

An evaluation of the teams’ performance (P4, Chapter 5) showed that the teams were not improving as expected by the decision makers; after two years the teams were still on average 3.7 times less productive and 2.3 times less autonomous than the target performance goals (the productivity and autonomy levels of the benchmark teams). Furthermore, no team managed to progress to a mature level (C or D). Thus, it was necessary to extend the time for the Indian site to take over the main responsibility regarding the product. Furthermore, the decision makers decided to prepare some developers from the Indian site to become software architects. The idea was to have people capable of providing on-site mentoring in India. To do so, the selected developers stayed in Sweden for six months and received specific training and mentoring.
Table 1.7: Classification at the relationship level for the example.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>AB</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Offshore</td>
<td>The target site is located in a different country.</td>
</tr>
<tr>
<td>Legal entity</td>
<td>Insourcing</td>
<td>Both sites are part of the same company (Ericsson)</td>
</tr>
<tr>
<td>Geographical distance</td>
<td>Far</td>
<td>The fastest flight between the two sites is longer than two hours.</td>
</tr>
<tr>
<td>Temporal distance</td>
<td>Small</td>
<td>There is a time zone difference of 3 hours and a half.</td>
</tr>
<tr>
<td>Software process distance</td>
<td>Equal</td>
<td>Both sites employ the same software workflows, role and practices.</td>
</tr>
<tr>
<td>Communication model</td>
<td>Balanced synchronicity</td>
<td>The communication between the sites is done via different types of media, such as Skype, email and videoconferencing.</td>
</tr>
</tbody>
</table>
The results of this thesis show that it is very challenging to handle onboarding in globally distributed legacy software projects. Learning and mentoring are challenging for both new developers and mentors in this context, because it is hard to transfer knowledge over large distances (see Chapters 4–6). This is amplified by the existence of large amounts of complex legacy code in this type of project, which is more difficult to learn. As a result, onboarding undertakings in the investigated context are associated with longer onboarding periods (see Chapters 4–6), bigger costs (see Chapters 4–6), lower predictability of the time required by new developers/teams to fulfill assigned tasks (see Chapter 7), and consequently more schedule and budget overruns. Additional time, cost, and uncertainty are mainly due to the following:

- When adding new sites in large-scale globally distributed legacy projects, remote mentoring is often employed due to the absence of personnel with enough legacy knowledge (see Chapters 4 and 5). This has an additional side effect, which is related to the allocation of expensive and scarce resources, such as software architects, to mentor newcomers. This means that these resources have limited time to embark on new projects and support innovation (see Chapter 4). Furthermore, their performance may be affected due to the time they have to allocate to support newcomers (see Chapter 6).
- It may be the case that the expectations/goals for new developers/teams are defined by decision makers located in far away countries (see Chapter 6). Thus, decision makers and newcomers may have different national cultures, which can lead to misunderstandings regarding the expectations/goals (see Chapter 5). Furthermore, communication and coordination are more difficult to handle in this context \[39, 79, 90, 112\], which can introduce “noise” when conveying the expectations/goals to developers/teams onboarded in offshore locations.
- Build trust, foster interpersonal relationships and information networks is also very important to be successful when onboarding newcomers [9]. However, existing literature [89, 92] and the findings of this thesis (see Chapters 5 and 6) show that it is more difficult to handle such things in globally distributed projects. While people can build trust, interpersonal relationships and information networks on the site level, it is hard to do the same with people located in sites located in other countries [150, 190].
- Globally distributed projects often involve multiple sites, which may have different national cultures. Considering that organizational cultures are influenced by national cultures [95], it is hard to create ex-
ately the same organizational culture in sites with different national cultures [164]. This may result in fragmented onboarding strategies within the same company (see Chapter 6) and different formality levels regarding onboarding. Since formal onboarding processes are seemed as more effective [9], the existence of different formality levels may lead to situations where a company is successful to onboard developers in a particular location and fails in others. While it is important to account for the particularities of involved locations (see Chapters 2 and 5), it is equally important to account for onboarding functions (tools, practices, methods, and techniques [9]) that are proved to be effective [9].

Strategies to mitigate the aforementioned issues range from foster temporary collocation (see Chapter 5) and visits of key stakeholders (see Chapter 6), to frequent meetings via video conferencing (see Chapter 5). Furthermore, the process put forward in this thesis can help companies to systematically account for the particularities of involved locations and the effective onboarding functions when strategizing onboarding of developers (see Section 9). This can help to avoid onboarding strategy fragmentation and different effectiveness levels among different sites.

10.1 Validity threats

The validity threats associated with each research paper included in this thesis are discussed in detail in each chapter. In this section, we provide a combined view on the validity threats related to the overall results of this thesis, which are discussed using the categories reliability, internal, construct and external validity described by Runeson and Höst [181].

Reliability

Reliability is related to the repeatability of a study, i.e. how dependent are the data and analysis on the involved researchers [181]. We minimized this threat in all chapters by involving several researchers in the design and execution of all studies. We also developed explicit protocols to conduct all studies.

In Chapters 2–7 company representatives verified our observations and findings to avoid false interpretations and inconsistencies. However, part of the collected data is qualitative and is highly dependent on the involved interviewees. Thus, it might, therefore, be very hard to obtain the same values with other informants. This means that the results based on qualitative data might be different with we had interviewed other interviewees. This may have impacted mainly: i) the identified learning challenges and the associated conclusions presented in Chapter 5; ii) the onboarding strate-
gies, recommendations and conclusions presented in Chapter 6; and iii) the conclusions presented in this chapter that are related to i and ii.

**Internal validity**

Internal validity is related to factors that researchers are unaware of or cannot control regarding their effect on the variables under investigation [181].

In Chapter 2, we made an attempt to account for as many confounding factors as possible by using existing literature and the knowledge of two experienced researchers (data and methodological triangulation). However, the resulting taxonomy might be different if the knowledge of more experts was accounted for when we developed the extended taxonomy.

In Chapters 3–7, we also made an attempt to account for as many confounding factors as possible through interviewing people with different roles and by using existing literature (data and methodological triangulation). Regarding the qualitative part of the case studies, the main internal validity threats are investigator bias and interviewee bias. To mitigate these threats, three researchers were involved with the design of the interview and workshop guides (investigator triangulation). We mitigated the second threat by interviewing people with different roles (data triangulation). It is important to mention that the set of investigated variables might be different if we had interviewed different interviewees, leading to different conclusions. However, we ensured that the involved interviewees had the appropriate knowledge to answer the designed questionnaires.

**Construct validity**

Construct validity reflects how well the measures used actually represent the constructs the study intends to measure [181]. This type of validity was threatened mainly in Chapters 3–7. The main threat to construct validity regarding these chapters is that we used only one method to measure each construct.

To mitigate the aforementioned threat in Chapters 3–7, we collected data from different sources and used different methods (data and methodological triangulations) to strengthen the produced evidence. Furthermore, we conducted a sanity check together with company representatives to validate the collected data.

**External validity**

External validity is concerned with the generalization of the findings [181]. Since the main research method employed in this thesis is case study, the main findings presented in this work are strongly bound by the context of the selected cases. The majority of the conducted case studies focused only
on one product from one company (the Ericsson case), which limits, even more, the generalization of the presented findings and solutions. Furthermore, the process proposed in this thesis was not empirically validated, although the solutions used as its build blocks were used in empirical studies.

Despite the aforementioned limitations, the main contributions of this thesis may be of interest and applicable to researchers and practitioners that work in a similar context: companies carrying out large-scale globally distributed projects with large amounts of complex legacy code.

11 CONCLUSIONS, IMPLICATIONS AND FUTURE DIRECTIONS

In this thesis, we holistically investigated challenges and implications associated with the onboarding of software developers and teams in large-scale globally distributed legacy projects (RQ1). We proposed a process to support the development of onboarding strategies and the evaluation of onboarding results (RQ2).

We identified the following challenges associated with the onboarding software developers and teams in large-scale globally distributed legacy projects:

1. The complexity of the product (complex legacy code) and technology stack.
2. The distance to the main source of product knowledge.
3. Lack of team stability.
4. Training expectation misalignment.
5. Lack of formalism and control over onboarding strategies employed in different sites of globally distributed projects.

The complexity of legacy code makes the learning process more difficult. Moreover, it is often amplified by the fact that mentors and the main source of knowledge are located in offshore locations. A lack of cultural awareness may impact the onboarding process as well if particularities of a given location are not accounted for. Finally, when many sites are involved, it is difficult to control how the steps of an onboarding undertaking will be planned and executed in all sites; there might be differences regarding organizational culture in different sites of large companies (like Ericsson).

We identified the following implications related to the onboarding software developers and teams in large-scale globally distributed legacy projects:

1. Longer times for software developers and teams to achieve high performance.
2. Higher cost for software developers and teams to achieve high performance.
3. Higher uncertainty regarding the effort required to fulfill tasks.
The results of this thesis indicate that onboarding in globally distributed legacy projects must be planned well ahead. It might take a long time for new developers and teams to acquire all knowledge required to perform as well as mature teams. Organizations must be prepared to provide and sustain extended periods of mentoring by expensive and potentially scarce resources (e.g., software architects). This may affect the effort necessary to carry out tasks and make it more difficult to plan schedules, which may lead to more frequent schedule overruns.

Regarding RQ2, we identified tools and practices and proposed a novel measurement approach and recommendations to support the onboarding of developers and teams. Furthermore, we combined the aforementioned solutions in the format of a process that can be used by companies to develop onboarding strategies and evaluate onboarding results. Companies can employ a subset of the process activities or the full process. The process can help companies to onboard developers and teams in a more effective, repeatable and systematic way.

As a result of this thesis, we identified the following research opportunities:

- The challenges and implications identified in this thesis are mainly related to one case from Ericsson. Thus, research and practice might benefit from the replication of our research in other large-scale globally distributed legacy projects.
- Considering that no other holistic investigation has been conducted regarding the onboarding of software developers/teams (accounting for all onboarding function categories [9] at the same time and their interplay), we believe that research and practice can benefit from more investigations in different contexts and domains using the approach employed in this thesis.
- Most research that focuses on the onboarding function categories has been conducted in the context of open source projects. This means that both research and practice can benefit from novel research conducted in the context of closed source projects, especially the ones with many people, a large amount of legacy code, and many sites.
- Although training is the most covered onboarding function category by literature, even having venues dedicated to it (e.g., IEEE Transactions on Education, Conference On Software Engineering, Education and Training (CSEE&T) and a special track in the International Conference on Software Engineering (ICSE)), there is no study that aggregates the existing empirical evidence. Secondary studies that aggregate empirical evidence could facilitate the conduction of onboarding related research and also make easier for practitioners to develop strategies to onboard software developers.
• It was not possible to evaluate in a quantitative way the effectiveness of the onboarding strategies employed in the investigated cases. However, it is important to do so to identify effective onboarding functions in a more accurate fashion and appropriate ways to combine them in different onboarding contexts [29]. Therefore, researchers should account for that when conducting onboarding related research.

• The empirical studies conducted to identify the challenges and implications covered approximately two years of data. We believe that investigations accounting for longer periods of time can support the identification of long-term challenges and implications related to the onboarding of software developers and teams in globally distributed legacy software projects.

• The solutions put forward in this thesis can help companies to strategize and evaluate the results of onboarding undertakings in a more formal, systematic and repeatable way. However, we believe that it is important to use them in other cases. In special, the proposed process has not been used to plan any onboarding undertaking from the scratch. Rather, its activities were used in an on-going undertaking (the Ericsson case). By evaluating our solutions in other cases, it will be possible to identify whether they fit different onboarding needs in other globally distributed legacy software projects.

Considering the aforementioned research opportunities, we plan to complement the research reported in this thesis by:

1. Collecting more data about the same case from Ericsson, to be able to identify long-term implications of onboarding on developers and teams in the target context;

2. Replicating the research presented in this thesis and use the proposed process in other cases, to increase the generalizability of our findings and solutions;

3. Quantitatively evaluating the effectiveness of the onboarding strategies employed in the three cases investigated in Chapter 6.
AN EXTENDED GLOBAL SOFTWARE ENGINEERING TAXONOMY

1  INTRODUCTION

1.1  Context

Throughout the years, the software industry has applied many different software development approaches, aiming at increasing process efficiency and profitability. Numerous companies worldwide develop software in a globally distributed manner (Global Software Engineering - GSE) to achieve benefits such as reduced time-to-market and access to skillful people all over the world [21, 49, 88, 172].

Despite all the benefits argued to be achieved by means of GSE, it comes with challenges. These challenges often impact the productivity [92] and effectiveness [68] of distributed software development, leading to delayed projects [90, 210].

The considerable number of delayed projects reported in literature indicates that practitioners have fallen short of providing accurate and reliable effort estimates in both collocated and globally distributed projects. To better understand these challenges, two studies related to effort estimation in GSE were carried out [24, 25]. These two studies among other findings confirmed the results reported by others, e.g. [214], i.e. that there is a lack of a common terminology in GSE, which makes it hard to compare and synthesize results across studies.

1.2  Problem outline

Britto et al. [24] conducted a systematic literature review (SLR) on effort estimation in the GSE context aiming at identifying the particularities of effort estimation in GSE projects. However, despite the relevance of this research topic, Britto et al. identified just a few studies supported by empirical evidence. In addition, the authors also found out that the related studies are reported in an ad-hoc manner, i.e. no common terminology
or knowledge organization scheme was used to report effort-related studies in the context of GSE projects. The absence of a common terminology and structured knowledge organization can hinder the understanding of studies’ contexts, making the studies harder to analyze and compare as well as aggregating the results from similar studies. Thus, it can hinder the advances in the field and the transfer of research results to industry. A classification scheme can mitigate the aforementioned problems [204].

In the context of GSE, it has been common to use taxonomies as classification schemes to organize the existing knowledge in the field [83, 129, 214]. According to the Oxford English Dictionary [166], a taxonomy is “a scheme of classification”. This concept was initially devised to classify organisms [132], although it has been applied in many different domains, e.g. education [16], psychology [151] and computer science [183].

Originally, the taxonomy approach was designed to classify knowledge in a hierarchical way. Nevertheless, to date many different classification structures have been used to construct taxonomies, e.g. “hierarchy”, “tree” and “facet-based” [127].

A classification scheme, such as a taxonomy, can be beneficial for both researchers and practitioners in four different ways:

1. It can ease the sharing of knowledge [204, 205, 220].
2. It can help to identify gaps in a particular knowledge area [204, 205, 220].
3. It can provide a better understanding of the interrelationships between the factors associated to a particular knowledge area [204].
4. It can support decision making processes [204].

1.3 Objective

Recently, Smite et al. proposed a GSE taxonomy, but their proposal does not include an exhaustive list of GSE factors; a taxonomy is a classification scheme that is expected to evolve over time [204]. Therefore, the main goal of this chapter is to extend Smite et al.’s taxonomy, adding new factors that were identified by means of two empirical studies, i.e. an SLR [24] and a survey [25].

1.4 Contribution

We achieved the objectives of this chapter with the following contributions:

• An extended GSE taxonomy based on evidence from an SLR and a survey.
• A validation of the proposed extended GSE taxonomy comparing it with other similar taxonomies and demonstrating its utility through classifying eight finished real GSE projects.

1.5 Outline

The remainder of this chapter is organized as follows. A discussion of the related work is presented in Section 2. Section 3 explains the applied research design and methodology. Section 4 presents the extended GSE taxonomy, followed by its validation in Section 5. Section 6 provides a discussion of the academic/industrial implications, which is followed by the discussion of the limitations of this work in Section 7. Finally, in Section 8 we draw our conclusions and present directions for future work.

2 BACKGROUND AND LITERATURE REVIEW

We identified three taxonomies [83, 129, 214] and two ontologies [141, 208] that provide knowledge organization schemes in the GSE context. It is important to note that other taxonomies were also identified [2, 94, 175], but they were already used as basis for Smite et al.’s taxonomy [214]. For this reason we decided not to consider them in this work. The taxonomy proposed by Narasipuram [159] was also not considered because its supporting evidence is limited.

The identified knowledge organization schemes can be categorized as follows:

• **Description approach**: Three studies [129, 141, 208] proposed graphical-based approaches that are more adequate to describe rather than to classify GSE projects. It comes from the fact that none of these approaches has dimensions with clear classification criteria associated to them; rather, they provide a set of “variables” that should be instantiated.

• **Classification approach**: The other two studies [83, 214] are more adequate to classify GSE projects, because they are organized in dimensions that have categories with associated classification criteria.

2.1 Smite et al.’s GSE taxonomy

Smite et al. [214] conducted a Delphi-inspired study with GSE researchers to develop an empirically based glossary and taxonomy, focused on the sourcing strategy aspect of GSE projects.

To construct the taxonomy, firstly, the authors investigated the state of the art in the use of GSE terminology by systematically reviewing stud-
ies from GSE-related venues. Secondly, by using a Delphi-based approach, they evaluated the literature and defined a consensual terminology. Finally, the authors identified the relationship between the defined terms using the defined terminology. To illustrate the usage of the proposed GSE taxonomy, the authors classified sourcing strategies presented in 68 different studies.

Figure 2.1: The GSE taxonomy (Adapted from Smite et al. [214]).

This taxonomy was developed to classify the relationship between pairs of sites, although it is equally possible to describe more complex GSE projects, with more than two sites. The taxonomy has five dimensions, as presented in Figure 2.1 and described in Table 2.1.

The taxonomy was designed using a facet-based classification structure [127]. It has five facets (dimensions), which relate to each other as follows:

- The dimension “GSE” is the parent of all the other dimensions.
- The classification by means of the dimension “geographic distance” depends on the categories of the dimension “location”.
- The classification by means of the dimension “temporal distance” depends on the categories of the dimension “geographic distance”.

50
Table 2.1: Dimensions of Smite et al.’s taxonomy.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSE</td>
<td>Sourcing</td>
<td>This dimension contains the root of the taxonomy, called sourcing. In this context, sourcing means some form of external software development.</td>
</tr>
<tr>
<td>Location</td>
<td>Onshore, Offshore</td>
<td>A sourcing can be delegated to a site in the same country, i.e. onshore, or to a site in another country, i.e. offshore.</td>
</tr>
<tr>
<td>Legal entity</td>
<td>Insourcing, Outsourcing</td>
<td>Independently from the location, a sourcing can be transferred to a different branch (site) of the company, i.e. insourcing, or subcontracted to a different legal entity (company), i.e. outsourcing.</td>
</tr>
<tr>
<td>Geographical</td>
<td>Close, Distant, Near, Far</td>
<td>The geographical distance is considered: <strong>distant</strong> when at least one flight is required to have face-to-face meetings, which yields time and cost increases. In offshore projects, the geographical distance is considered: <strong>near</strong> when the required flying time is less than two hours; <strong>far</strong> when the flying time is longer than two hours and staying overnight is usually required.</td>
</tr>
<tr>
<td>distance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temporal distance</td>
<td>Similar, Different, Small, Large</td>
<td>In onshore projects, the temporal distance is considered: <strong>similar</strong> when there is a time difference of one hour or less; <strong>different</strong> when the time difference between two sites is longer than one hour. In offshore projects, the temporal distance is considered: <strong>small</strong> when there is a time distance between sites of four hours or less; <strong>large</strong> when there is a time distance between two sites of more than four hours.</td>
</tr>
</tbody>
</table>

Gumm [83] developed a taxonomy to classify GSE projects in terms of distribution dimensions. Its goal was to provide a foundation to discuss the challenges related to GSE projects and was based on an earlier literature study performed by the same author.

The proposed taxonomy uses four different dimensions (physical distribution, organizational distribution, temporal distribution and distribution between stakeholder groups) to classify the ways in which people and artifacts can be distributed in GSE projects. Each dimension can be measured on a 3-point ordinal scale (high, medium or low). The author describes an onshore distributed project to validate her proposal and she argues that the taxonomy helps to understand the scope and the distribution issues of the evaluated project.

Laurent et al. [129] proposed a taxonomy and a visual notation to address the requirements engineering aspect of GSE projects. These authors’ main goal was to design a common language for modeling the requirements of GSE projects and to allow project managers to manage distributed requirements in a better way. The proposal was derived from the findings of a broad study performed with industrial partners (seven different projects). Interviews were performed with the team leaders responsible for eliciting and gathering the requirements in each project.

The taxonomy is divided into three different entities: role, site and artifact. They graphically showed the taxonomy as a Unified Model Language (UML) class diagram with some attributes in each entity. These attributes are related to the entity “site” and respectively called location, language and time zone.

To facilitate the taxonomy’s usage, the authors also designed a visual notation, which was later on used to describe a real GSD project from the video gaming domain. They report that the taxonomy help to identify problems in advance regarding the management of documents and the requirements gathering process.

Vizcaino et al. [208] developed an ontology, called O-GSD, which was aimed at easing the communication and avoiding misunderstanding in GSE projects. This ontology was iteratively developed in the context of a project that involved five companies and two universities in Spain. The authors used the REFSEN0 (representation formalism for software engineering) [196] to create the ontology.

The ontology allows for the description of GSE projects by the instantiation of different factors, e.g. time zone difference and language distance, roles of the involved members and involved sites. The authors designed a glossary and a UML class diagram to depict the semantic relationship between all the determined concepts.
To validate their proposal, the ontology was used to describe a real GSE project, which consisted of software related to the sale of security devices in the European Union countries. The ontology was able to cover all the concepts required by the involved company to represent the GSE project, and in addition it also fostered a common understanding about the represented project.

Marques et al. [141] introduced an ontology for team task allocation in GSE projects. This ontology was developed based on the findings of a systematic mapping study performed by the authors and aimed at clarifying the concepts related to team task allocation in distributed projects.

The authors used UML class diagrams to represent the ontology. The main concepts addressed by the authors were artifact, competence and constraints. The authors performed a preliminary evaluation of the ontology by interviewing five project managers, and the results suggested that the concepts and relationships embraced by the ontology were suitable to be applied in real distributed projects.

2.3 Research gaps

Only the taxonomies proposed by Gumm [83] and Smite et al. (base GSE taxonomy) [214] are considered as knowledge classification approaches. The base GSE taxonomy is more comprehensive, providing a wider range of relevant dimensions and clear criteria to classify GSE projects. In addition, this taxonomy was also developed with the participation of several GSE experts, which provides the taxonomy more strength and credibility.

Despite the usefulness of the base taxonomy, to classify GSE projects in a more comprehensive way, additional aspects must be considered (as identified in our previous work [24, 25]):

- It is well-known by the GSE community that language and cultural factors have an important role in GSE projects [24, 25, 49, 88, 90]. Despite both factors being discussed by Smite et al. [214], their taxonomy does not have dimensions to represent these factors.
- The base taxonomy was developed to classify relationships between pairs of sites. Nevertheless, some GSE factors would be better classified in the granularity level of site rather than the granularity level of relationship-between-a-pair-of-sites, e.g. a site’s software process type and cultural factors.

3 Research design and methodology

This section presents the research design and methodology used herein. The following research questions drove the work reported in this chapter:
• RQ1: What dimensions are needed to augment the usefulness of Smite et al.’s taxonomy?
• RQ2: What is the utility of the extended taxonomy?

To answer RQ1 and RQ2, we followed the process presented in Figure 2.2.

First, we identified the dimensions to be incorporated into the original taxonomy. The results of one SLR [24] and one survey [25] were used as input to this step. Only the dimensions reported in more than one primary study in the SLR and later on confirmed by the survey were selected, i.e. only the dimensions with empirical evidence were considered. In this step, we identified four new dimensions (software process model, cultural factors, language and communication model) not present in the original taxonomy, but judged as essential to capture in the taxonomy.

Second, we identified categories for each dimension. To do so, we used relevant literature related to each dimension (see Section 4) and our own knowledge to identify meaningful categories with clear classification criteria. Clear classification criteria facilitate the usage of the taxonomy, and help in making correct classifications of the subject matter [216].

During this step, we identified the need to split the dimensions related to culture and software process, as follows:

• “Culture” was split into two dimensions: “power distance” and “uncertainty avoidance”.
• “Software process” was split into two dimensions: “software process type” and “software process distance”.

We did so to enable our extended taxonomy to classify culture and software process related factors on a finer grained level, which we believe

---

1 In this chapter, the subject matter to be classified is a GSE project.
would enhance its usefulness and enable the classification of a wider range of GSE contexts.

Third, we combined the new dimensions with the dimensions of the original taxonomy. In doing so, we identified some inconsistencies in the resulting extended taxonomy; most new dimensions are site-related, but the original taxonomy was designed to classify only relationships between pair of sites (see Figure 2.3 in Section 4).

Fourth, to address this inconsistency, we added one new dimension, called “setting”, which enables the classification of GSE projects in both site level and relationship-between-pair-of-sites level. We also adjusted the original dimension “GSE” to keep consistency, i.e. its category was renamed to project.

Fifth, we validated our extended taxonomy. A taxonomy can be validated in three ways [214]:

- **Orthogonality demonstration** - The orthogonality of the taxonomy dimensions and categories should be demonstrated.
- **Benchmarking** - The taxonomy should be compared with other similar classification schemes.
- **Utility demonstration** - The utility of the taxonomy should be demonstrated through the classification of existing knowledge.

The orthogonality of the new dimensions was ensured by defining categories with clear classification criteria (see Section 4).

The benchmarking was carried out by comparing the extended taxonomy with the base taxonomy [214] and Gumm’s [83] proposals. We used only these two taxonomies to perform the benchmarking because the other knowledge organization schemes did not provide clear criteria to perform knowledge classification, as discussed in Section 2. Our benchmarking is further detailed in Section 5.2.

Finally, to demonstrate the utility of our extended taxonomy, we illustrate its usage by classifying eight real finished GSE projects. This illustration is presented in Section 5.1.

4 THE EXTENDED GSE TAXONOMY

Figure 2.3 displays the extended GSE taxonomy proposed herein. Seven new dimensions were incorporated into the base taxonomy: “setting”, “software process type”, “software process distance”, “power distance”, “uncertainty avoidance”, “language distance” and “communication model”.

The relationship between the new and original dimensions are as follows:

---

2 The details of the original dimensions are not shown to facilitate the presentation of the new dimensions. Figure 2.1 illustrates the original dimensions.
Figure 2.3: Extended GSE taxonomy.
• The dimension “GSE” is the parent of all the other dimensions.
• The classifications by means of the dimensions “software process type”, “power distance”, “uncertainty avoidance” and “language distance” are related to the category site of the dimension “setting”.
• Classifications by means of the dimensions “software process distance” and “communication model” are related to the category relationship of the dimension “setting”.

The seven new dimension are further detailed in Sections 4.1 (GSE), 4.2 (setting), 4.3 (software process type and software process distance), 4.4 (power distance and uncertainty avoidance), 4.5 (language distance) and 4.6 (communication model) respectively.

4.1 GSE

GSE is the root of the taxonomy. To better manage to classify on both site and relationship-between-pair-of-sites granularity levels, project has been introduced into the root level instead of sourcing, as proposed in the original taxonomy. Herein we consider a project as “a temporary endeavor undertaken to create a unique product, service, or result” [101].

4.2 Setting

The dimensions of the extended taxonomy are formulated to classify GSE projects on both the site level (Site) and the relationship-between-pair-of-sites level (Relationship).

A site is defined as a unit composed of human resources that interact with other sites (nodes). We define a relationship as the relationship between two sites interacting in a project (edge).

4.3 Software process dimensions

The software development process type used (agile [179], plan-driven [188] or hybrid [126, 207]) is an aspect that impacts the conduct of a GSE project, e.g. the effort required to perform such projects [24, 25]. In addition, the way the practices are incorporated into the sites’ routines can also be different (workflows). Differences between software processes used in different sites may lead to problems in the communication and loss of trust [172], for example impacting the associated effort [24, 25].

Therefore, we incorporated the dimensions software process type and software process distance to account for software process factors.
Plan-driven software development may be viewed as heavy and bureaucratic to deal with certain types of projects, specially the ones where the requirements are unclear and uncertain [75]. Therefore, the main criticism regarding plan-driven development is that many decisions that are taken early on must be reappraised later on, since software development deals with a lot of uncertainty in the early stages of a project [169]. Nevertheless, this approach allows for planning organizational aspects earlier, besides fostering the discovery of potential problems before the start of a particular project.

Agile methods are regarded as being more suitable to deal with projects that present unclear and uncertain requirements, but they demand close collaboration between the customer and the development team [14]. Furthermore, organizations and customers may be more familiar with plan-driven approaches and may find it hard to trust and follow an agile-based approach [78]. Pure agile-based software processes are difficult to scale. They are more adequate to small and medium size projects [78]. Finally, existing empirical evidence suggests that agile practices are not readily applicable to GSE projects [98, 107].

A software process at the project level may be split, which enables distributed teams to combine practices from both agile and plan-driven approaches, and hence generating software process diversity [173]. Software diversity can help teams to address the limitations of pure agile or plan-driven software processes by combining the practices from each approach that fits each case [173], thus leading to hybrid processes [126, 207]. For example, some organizations have a more plan-driven mentality about project management practices, but the teams may still be mainly agile.

Considering the discussion above, we define the dimension “software process type” as having two categories:

- **Agile** - The software process used in a site is mainly based on agile practices.
- **Plan-driven** - The software process used in a site is mainly based on plan-driven practices.

We did not include a category **hybrid** in this dimension, because one of the types of practice (agile or plan-driven) is expected to be more prevalent. Furthermore, most organizations would probably perceive themselves as using a hybrid approach, and hence it is viewed as more important to know the process type being most commonly used with respect to the objective of a study. For example, an organization may have agile teams that are managed in a more plan-driven way; if the main focus of the classification is the effort associated with the software development in each site, the best classification would be **agile**, because the teams use mainly agile prac-
tices to develop software; however, if the main focus of the classification is the effort associated with coordination between sites, *plan-driven* would fit the best, since management is more plan-driven than agile.

**Software process distance**

While software diversity can help teams to overcome the limitations associated with “pure” software processes (pure agile or pure plan-driven), it may result in differences between the software processes of different sites. To account for this, we incorporated the dimension *software process distance*, which enables the classification of the distance between two sites in terms of the software processes used. This dimension has the following categories:

- **Equal** - The software processes of the sites are very similar, i.e. they use the same workflows, roles and practices to develop software.
- **Similar** - The workflows, roles and practices are not the same in both sites, but the sites use software processes that are based on the same type of software development practices (mainly agile or mainly plan-driven).
- **Different** - The sites neither use the same type of software development practices nor use the same workflows, roles and practices.

4.4 **Cultural factors**

Both national and organizational cultures influence both decision-making and the way development is conducted in a project. In GSE projects, the different cultures involved can impact negatively on the communication and trust between sites [54], and can lead, for example, to a bigger effort [24, 25].

Culture is represented in our extended taxonomy using Hofstede’s national culture framework [95]. From Hofstede’s framework, we have only adopted two dimensions that are named *power distance index* - PDI and *uncertainty avoidance index* - UAI. They have been adopted because empirical evidence exists for these two dimension; the evidence supports their influence on the organizational level [95], which is the level in which projects are carried out.

Hofstede’s PDI and UAI are defined as follows:

- **Power distance index (PDI)** - Measures how people manage inequality in hierarchical relationships, i.e. manager-subordinates. In nations with high PDI, the employees depend more on the managers to make decisions. However, in nations with low PDI, the competences of the employees are higher valued than their hierarchical position.
• **Uncertainty avoidance index (UAI)** - Measures how people manage uncertainty, how they feel threatened by uncertain situations and try to avoid or mitigate such situations. In nations with strong uncertainty avoidance, they have strict laws and rules. Nevertheless, nations with weak uncertainty avoidance have as few rules as possible, which make their people more tolerant to uncertain situations.

Based on Hofstede’s framework, we designed the dimensions *power distance* and *uncertainty avoidance* to account for the cultural factors in our extended taxonomy.

*Power distance*

PDI is represented in our extended taxonomy by the dimension called *power distance* (PD), which has the following categories:

- **Small** - Sites placed in countries with $\text{PDI} \leq 50$ have small power distance.
- **Large** - Sites placed in countries with $\text{PDI} > 50$ have large power distance.

*Uncertainty avoidance*

In our extended taxonomy, UAI is represented by the dimension called “uncertainty avoidance” (UA), which has the following categories:

- **Weak** - Sites placed in countries with $\text{UAI} \leq 63$ have weak uncertainty avoidance.
- **Strong** - Sites placed in countries with $\text{UAI} > 63$ have strong uncertainty avoidance.

The threshold values used to differentiate sites with Small or Large PD and Weak or Strong UA were defined based on Hofstede et al.’s empirical study [95] [3]. To choose the proper UA and PD categories for a site, UAI and PDI scores for the countries involved in a project under classification should be determined; the scores are available in Hofstede et al.’s book [95].

Note that outcomes of the PD and UA classifications for sites should be compared. For example, consider a project with two sites, respectively named X and Y. Site X is placed in Germany and site Y is placed in Brazil, i.e. X’s PD is Small and UA is Strong, while Y’s PD is Large and UA is Strong. In this example, there is no major concern about the impact of UA on the GSE project, since both sites are classified in the same category. However,

---

[3] Hofstede et al.’s [95] provide a table and a figure that contains the UAI and PDI of many different countries.
PD could negatively impact the GSE project, since the sites are classified in different categories.

In some situations, companies source human resources from different countries to compose teams in one location. This means that the main national culture of a particular site is not necessarily the national culture of the country wherein the site is placed. For example, Ramasubbu and Balan [170] report a project with two sites, one placed in the USA and the other one located in India. Although both sites are placed in countries with different PD and UA, the human resources of both sites were from India. In such situation, the actual cultural distance between the two sites is expected to be zero.

Therefore, it is important to account for the predominant nationality of the human resources of a site to define the appropriate PD and UA.

4.5 Language distance

In a GSE project, it is very likely that involved sites do not have the same native language, which may lead to misunderstandings between sites and generate delays in the entire project [3]. Nowadays, English is the most commonly used language when there is need for a lingua franca [135]. Thus, instead of calculating the distance between sites’ languages, we incorporated a dimension named language distance, which classifies the distance between each site’s language and English.

This dimension has the following categories:

- **No distance** - When the mother language of a site is English, or no lingua franca is required. In the latter case there is no language distance in such a site, since people from both sites could communicate in their native tongue.
- **Small** - When $0 < L_d \leq 0.4$, the language distance of a site is considered small. This means that it is more likely that people from such a site have an acceptable level of proficiency in English, since it is relatively easy for them to learn it.
- **Medium** - When $0.4 < L_d \leq 0.57$, the language distance of a site is considered medium. This means that it is more likely that people from such a site struggle somewhat to learn English, which affects their proficiency. However, they can learn and speak English by applying more effort than people from the previous group.
- **Large** - When $0.57 < L_d \leq 1$, the language distance of a site is considered large. This means that it is more likely that people from such a site struggle even more to learn English. In general, those languages have almost no commonalities with English, which requires more effort to learn English.
In the aforementioned categories, Ld represents the distance between the language of a particular site and English [44]. According to Chiswick et al., Ld can assume the following values: 0.33, 0.36, 0.4, 0.44, 0.5, 0.57, 0.67, 0.8 and 1. The bigger the Ld value, the farther a particular language L is from English, which is also a measure of how difficult it is for people who speak L to learn to speak English. Thus, the larger the Ld, the higher the likelihood that the proficiency in English will not be very good [44]. The lower the level of proficiency in English (as lingua franca), the higher the probability of problems regarding the communication between sites [88].

Note that this dimension of our taxonomy can be used only in projects that require no lingua franca to enable communication between sites (i.e. there is no language distance) or when the chosen lingua franca is English.

The first category of this dimension (No distance) was designed to represent sites that either have English as its mother tongue or no lingua franca is required in the project, since the sites have the same mother tongue. The other three categories were defined by dividing the language distance scale in three equal parts (Small, Medium and Large), so that there is enough representativeness to classify the existing spectrum of language distance values.

This dimension focuses on the language that is spoken the most in a site’s location. We did so because it would be very difficult to embrace the particularities of countries that have more than one official language. In addition, high proficiency in English is a prerequisite to allocate personnel to participate in many GSE projects, i.e. in these cases, the mother tongues of sites’ locations are not an issue.

Thus, when using this dimension to classify the language distance of each site, the language spoken the most by the site’s personnel should be identified and it should be used as basis for selecting the language distance category that fit the best.

4.6 Communication model

In GSE projects, the communication between the distributed sites is often mediated via electronic communication media [104] and existing empirical evidence shows that mediated communication demands more effort [55, 63]. Different electronic communication media types have different properties and capabilities to deal with geographic distance between sites in GSE projects.

Media synchronicity theory (MST) [56] states that the most effective communication occurs when the communication media matches a given set of communication requirements [56]. The information to be transmitted can

---

4 Chiswick et al. [44] provide a table that contains the Ld of many different countries.
require more conveyance, i.e. processing and transmission of new information, or more convergence, i.e. a group should agree on something [56].

In our extended taxonomy, we incorporated the communication factor through the dimension called communication model, which is based on the MST. The communication model dimension has the following categories:

- **Low synchronicity communication model** - The communication between sites is mainly mediated via asynchronous media, e.g. email and issue trackers (the most adequate media type for conveyance [56]).
- **High synchronicity communication model** - The communication between sites is mainly mediated via synchronous media, e.g. video conference and instant messaging tools (the most adequate media type to achieve consensus [56]).
- **Balanced synchronicity communication model** - The communication model encompasses both asynchronous and synchronous media types and each type is used for its most adequate purpose, i.e. synchronous media is used when there is need for fast feedback and consensus achievement, and asynchronous media is used when there is need to convey some message that should be formalized and consolidated before transmission.

5 Validation

A taxonomy can be validated through orthogonality demonstration, benchmarking and utility demonstration. We ensured the orthogonality of the extension’s dimensions by defining categories with clear classification criteria (see Section 4). We demonstrate the utility of the extension in Section 5.1 and a benchmarking is presented in Section 5.2.

5.1 Demonstration of utility

To demonstrate the utility of our extended taxonomy, we classified eight finished real GSE projects. These projects were obtained from Ramasubbu et al. ([171]). Since not all the projects’ data required to perform the classification was available in the paper, we contacted the authors to complement the missing data. We provided them an excel spreadsheet with eight tabs (one per project). Each tab had the following fields: project’s ID (PID), company’s name, software domain, site’s ID (SID), site’s country, site’s city, site’s main language, software process type, software process distance, communication model, legal entity, relationship ID (RID) and relationship’s sites. Clarifications related to the spreadsheet was provided whenever required.
Due to a non-disclosure agreement, the name of the companies and the domain of the software applications developed/maintained were not provided. The cities wherein the sites were placed were also not explicitly stated, also because of the non-disclosure agreement. Rather, they presented the location of the sites in terms of countries. In the case of sites placed in the USA, the region (e.g. north, south, east, west) of each site was also provided. In addition, the authors also clarified that the team members of all projects were fluent in English. Thus, we considered that there was no language distance between the sites of the projects.

We used the provided data to conduct the classification of the eight projects. Figure 2.4 shows the setup of each project, i.e. the connections between the involved sites 5, while Tables 2.2 and 2.3 6 show the classification for each project in the site level, and Tables 2.4 and 2.5 show the classification of each project in the relationship-between-pair-of-sites level.

The projects classified by means of the proposed extended dimension provided a variety of different setups:

- Project 1 had seven sites and project management was concentrated in site A (USA), i.e. there was no interaction between sites B, C, D, E, F and G; all the sites only interact with A.
- In Project 2, four onshore sites (USA) were involved and all the sites directly interact with each other, although site A had the biggest responsibility regarding project management.
- Project 3 had four sites and project management was concentrated in site A (India). Despite the fact that sites B, C and D were all located in the USA, they did not interact with each other, only with site A.
- In Project 4, four sites were involved and project management was concentrated in site A (India). The other sites only interact with site A.
- Project 5 and project 6 had two sites involved each and project management was mainly the responsibility of site A for both project 5 and project 6 (India and Germany respectively).
- In Project 7, four onshore sites (India) were involved and all the sites directly interact with each other, although site A had the main responsibility regarding project management.
- Project 8 had four sites and project management was concentrated in site A (USA). Despite the fact that sites B, C and D were all located in India, they did not interact with each other, only with site A.

The classification results can be used in many different ways, such as:

---

5 The nodes represent the sites and the edges represent the relationships between sites.
6 Due to the fact that there was no language distance in any of the projects, we omitted this dimension in Tables 2.2 and 2.3.
Figure 2.4: Setup of the classified projects.
Table 2.2: Classification in the site level – part I.

<table>
<thead>
<tr>
<th>PID</th>
<th>SID</th>
<th>Country</th>
<th>Software process type</th>
<th>Power distance</th>
<th>Uncertainty avoidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>1</td>
<td>E</td>
<td>Singapore</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>Germany</td>
<td>Agile</td>
<td>Small</td>
<td>Strong</td>
</tr>
<tr>
<td>1</td>
<td>G</td>
<td>Australia</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>2</td>
<td>D</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>3</td>
<td>B</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>4</td>
<td>B</td>
<td>Germany</td>
<td>Agile</td>
<td>Small</td>
<td>Strong</td>
</tr>
<tr>
<td>4</td>
<td>C</td>
<td>Spain</td>
<td>Agile</td>
<td>Small</td>
<td>Strong</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>England</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>Japan</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Strong</td>
</tr>
</tbody>
</table>
Table 2.3: Classification in the site level – part II.

<table>
<thead>
<tr>
<th>PID</th>
<th>SID</th>
<th>Country</th>
<th>Software process type</th>
<th>Power distance</th>
<th>Uncertainty avoidance</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>A</td>
<td>Germany</td>
<td>Agile</td>
<td>Small</td>
<td>Strong</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>India</td>
<td>Agile</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>7</td>
<td>B</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>7</td>
<td>D</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>USA</td>
<td>Agile</td>
<td>Small</td>
<td>Weak</td>
</tr>
<tr>
<td>8</td>
<td>B</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>8</td>
<td>C</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
<tr>
<td>8</td>
<td>D</td>
<td>India</td>
<td>Plan-driven</td>
<td>Large</td>
<td>Weak</td>
</tr>
</tbody>
</table>
Table 2.4: Classification in the relationship-between-pair-of-sites level – part I.

<table>
<thead>
<tr>
<th>PID</th>
<th>RID</th>
<th>Process</th>
<th>Communication</th>
<th>Location</th>
<th>Legal entity</th>
<th>Geographic</th>
<th>Temporal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AB</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Distant</td>
<td>Different</td>
</tr>
<tr>
<td>1</td>
<td>AC</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>1</td>
<td>AE</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>1</td>
<td>AF</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>1</td>
<td>AG</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>2</td>
<td>AB</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Similar</td>
</tr>
<tr>
<td>2</td>
<td>AC</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Different</td>
</tr>
<tr>
<td>2</td>
<td>AD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Different</td>
</tr>
<tr>
<td>2</td>
<td>BC</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Similar</td>
</tr>
<tr>
<td>2</td>
<td>BD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Different</td>
</tr>
<tr>
<td>2</td>
<td>CD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Outsourcing</td>
<td>Distant</td>
<td>Similar</td>
</tr>
<tr>
<td>3</td>
<td>AB</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>3</td>
<td>AC</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>3</td>
<td>AD</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
</tbody>
</table>
Table 2.5: Classification in the relationship-between-pair-of-sites level – part II.

<table>
<thead>
<tr>
<th>PID</th>
<th>RID</th>
<th>Process</th>
<th>Communication</th>
<th>Location</th>
<th>Legal entity</th>
<th>Geographic</th>
<th>Temporal</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>AB</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>4</td>
<td>AC</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>4</td>
<td>AD</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>5</td>
<td>AB</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>6</td>
<td>AB</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Insourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>7</td>
<td>AB</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>7</td>
<td>AC</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>7</td>
<td>AD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>7</td>
<td>BC</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>7</td>
<td>BD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>7</td>
<td>CD</td>
<td>Similar</td>
<td>Balanced synchronicity</td>
<td>On-shore</td>
<td>Insourcing</td>
<td>Dist-tant</td>
<td>Similar</td>
</tr>
<tr>
<td>8</td>
<td>AB</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Outsourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>8</td>
<td>AC</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Outsourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
<tr>
<td>8</td>
<td>AD</td>
<td>Different</td>
<td>Balanced synchronicity</td>
<td>Off-shore</td>
<td>Outsourcing</td>
<td>Far</td>
<td>Large</td>
</tr>
</tbody>
</table>
• To help researchers to classify studies so that other researchers and practitioners can more easily identify cases of particular interest to them. This directly relates to the next item.
• To identify cases of interest in the literature; the extended taxonomy provides a richer classification than the existing taxonomies. Thus, it improves the identification of similar cases in the literature. For example, if a practitioner or researcher intends to find literature related to GSE projects with sites that use different software processes, it would be easier to identify relevant studies if researchers report the studies using the extended taxonomy.
• To identify relationships between the classification results associated with each dimension and some other factor, which eventually can support decision-making processes. For example, it would be possible to use the classification results along with the effort associated with software development projects to perform a regression analysis and identify the relationship between the factors in the classification and the effort of GSE projects. The regression analysis result could be used to define which setup would be more adequate for a particular situation. The extended taxonomy allows for identifying relationships that involves software process-related aspects, sociocultural/language aspects, communication aspects and site-related aspects.

5.2 Benchmarking

To further validate and justify the need for the extended taxonomy presented herein, we compared the extension with the base taxonomy [214] and Gumm’s [83] taxonomy. We performed the comparison considering the following aspects:

• The classification results presented in Section 5.1.
• The taxonomy’s basis.
• Type of classification structure used to design the taxonomy, which can be hierarchy, tree, paradigm and facet-based [127].
• Procedure used to classify GSE projects, which can be qualitative or quantitative [216].
• Description of the dimensions’ categories (classification criteria), which can be objectively or subjectively described.
• Validation approach.

Tables 2.6 and 2.7 display the values related to the aforementioned factors, which are further elaborated next.
Table 2.6: Comparison between the extended taxonomy, the base taxonomy and Gumm’s taxonomy – part II.

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Basis</th>
<th>Dimensions</th>
<th>Structure</th>
<th>Procedure</th>
<th>Description</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended</td>
<td>Existing taxonomy, Systematic literature review and survey findings, and expert knowledge</td>
<td>12 (all the identified dimensions)</td>
<td>Facet-based</td>
<td>Qualitative</td>
<td>Objectively described</td>
<td>Comparison with existing taxonomies and demonstration of utility using data from eight finished GSE projects.</td>
</tr>
</tbody>
</table>
Table 2.7: Comparison between the extended taxonomy, the base taxonomy and Gumm's taxonomy – part II.

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Basis</th>
<th>Dimensions</th>
<th>Structure</th>
<th>Procedure</th>
<th>Description</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>Existing taxonomies, literature review and expert knowledge</td>
<td>5 (GSE, location, legal entity, geographic distance and temporal distance)</td>
<td>Facet-based</td>
<td>Qualitative</td>
<td>Objectively described</td>
<td>Based on expert judgment, comparison with existing taxonomies and demonstration of utility by classifying GSE literature.</td>
</tr>
<tr>
<td>Gumm's</td>
<td>literature review</td>
<td>3 (legal entity, geographic distance and temporal distance)</td>
<td>Facet-based</td>
<td>Qualitative</td>
<td>Subjectively described</td>
<td>Demonstration of utility using data from one finished GSE project.</td>
</tr>
</tbody>
</table>
Classification results

Considering the base taxonomy’s lack of certain dimensions as described in Section 2, including not covering software process-related aspects, socio-cultural/language-related aspects, communication-related aspects and site-level aspects, the extended taxonomy presented herein adds new dimensions that are believed to be important to better understand the context of GSE projects. Thus, our extended taxonomy can simplify the comparison between different studies, in particular when classifying new studies, so making it easier for both researchers and practitioners to compare their new cases with the existing literature on the topic.

When analyzing the classification results presented in Tables 2.2, 2.3, and 2.4, the base taxonomy classification coverage is equal to 42% (lack of seven dimensions) when compared to the extended taxonomy, while Gumm’s taxonomy is able to cover only 25% (lack of nine dimensions). This means that the dimensions that are relevant to describe GSE research contexts in a more comprehensive way are only covered by the extended taxonomy. In summary, the base taxonomy and the taxonomy by Gumm are subsets of the extended taxonomy.

Basis

The base taxonomy is an empirically based taxonomy, i.e. it was designed based on existing literature and also on the knowledge of several experts. Our extended taxonomy is based on the base taxonomy, two systematic empirical studies and expert knowledge. Gumm’s taxonomy was based on a non-systematic literature review. Thus, Gumm’s taxonomy lacks the empirical basis as compared to our extended taxonomy and the base taxonomy.

Classification structure

All three taxonomies were designed using faceted analysis [127] (facet-based classification structure), i.e. that GSE projects are classified along different perspectives (dimensions). It is not surprising that the three taxonomies were designed via this approach, because faceted analysis is the most adequate approach to classify knowledge of new and evolving knowledge areas [127], which is the case of GSE.

Classification procedure

Classification procedures define how subject matter instances (GSE projects) are systematically assigned to the categories of each dimension [216]. Qualitative classification procedures are based on scales while quantitative clas-
sification procedures are based on ratio scales [216]. The three taxonomies use nominal scales.

Classification criteria description

Both our extension and the base taxonomy provide very clear and objective descriptions for all their dimensions’ categories, i.e. the classification criteria are clear. Gumm does not provide a very clear description of its taxonomy’s categories, which makes it harder to use her taxonomy when compared to the usage of the other two taxonomies.

Validation

As discussed in Section 3, a taxonomy can be validated through orthogonality demonstration, benchmarking and utility demonstration.

The dimensions’ orthogonality in both our proposal and the base taxonomy was ensured by defining categories with clear classification criteria (see Section 4). Gumm does not discuss or demonstrate the orthogonality of her taxonomy’s dimensions.

The benchmarking was done by comparing the extended taxonomy with the two other taxonomies, i.e. Smite et al.’s and Gumm’s taxonomies respectively.

The utility of each one of the three proposals was demonstrated by using each taxonomy to perform actual classification. We did so by classifying eight real finished GSE projects whose data was provided by a GSE expert. Smite et al. classified the sourcing strategy of projects reported in 296 research papers. They also provided an example to explain how the taxonomy can be used to identify literature related to specific settings. Gumm used her taxonomy to classify one finished real GSE project.

In our case, an expert provided all the data required to perform the classification. In Smite et al.’s case, they extracted the required information from the classified papers. In some cases they had to infer the data from the text, since the required data was not clear up front; however, this was intentional to illustrate how the data in many papers is insufficient to classify studies of GSE. Gumm does not explicitly explain how the required data to perform the classification was obtained.

6 Discussion

Looking at the gaps (absent dimensions) associated with the base taxonomy, the existing literature suggests their relevance, and hence supports the need for a classification scheme that addresses them. This allows both researchers and practitioners to identify the most relevant cases in the literature in relation to their respective contexts. Abufardeh and Magel [1]
showed that the research addressing the impact of the cultural and linguistic aspects of software developed in a global distributed manner is limited. Mishra and Mishra [148] have conducted a literature review on cultural issues in GSE. They noted that the findings in the primary studies were not reported in a standard way. Ramasubbu et al. [173] have recently brought up the issue of software diversity, which has a significant impact on globally distributed projects and is to be yet further investigated. Jaanu et al. [104] have highlighted the importance of the type of media for effective communication in GSE projects, although GSE researchers are not reporting this aspect very often.

Our extended taxonomy can draw the attention of GSE researchers about the aforementioned aspects and help them to report these aspects in a more comparable way. It has also the potential to foster new research, specifically for the areas that are less covered by the existing literature (e.g. software process diversity.)

To support the research community to “speak the same language”, it is important to ensure that there is a consensual terminology and that this consensual terminology will not fragment over time. This work illustrates that it is possible to keep evolving a taxonomy of general use without fragmenting its content. It is important to emphasize that the extended taxonomy is fully consistent with the base taxonomy and can be used to classify GSE projects in any type of GSE study. Although the need for an extended taxonomy came from two studies related to effort estimation in GSE [24, 25], its usage is not limited to effort-related studies.

The extended taxonomy can help GSE researchers to report the context of new GSE research in a more systematic, clear and comparable way. Therefore, it can facilitate the analysis, comparison and aggregation of results from new studies, fostering advances in the knowledge field. Once new studies are reported in a more standardized manner, researchers will also be able to spend less effort to find relevant literature whenever necessary. The use of the extended taxonomy to report GSE studies can help practitioners to identify useful literature related to different contexts and consequently also helping them to address different problems whenever required.

In summary, the extended taxonomy allows for the aforementioned benefits by complementing existing taxonomies when it comes to the classification of GSE projects accounting for software process, socio-cultural, language and communication related factors, in both site and relationship-between-pair-of-sites granularity levels.
As with most studies, the research reported herein comes with some limitations. First, the dimensions of our extended taxonomy, both the new ones based on the two empirical studies by Britto et al. [24, 25] and the original dimensions by Smite et al. [214], do not represent an exhaustive list. However, the taxonomy can be further extended new factors are identified. Furthermore, it can be specialized by the incorporation of factors that are of interest for only a particular perspective (e.g. effort estimation). Thus, we encourage other GSE researchers to look at GSE projects from other perspectives (e.g. coordination, testing and effort estimation) to identify potential new dimensions that can augment the representativeness of our proposal.

The extended taxonomy has not yet been used by practitioners in ongoing projects, i.e. there is no empirical evidence ensuring the utility of our proposal in this kind of situation. So, we also encourage GSE researchers to conduct studies in industrial settings to strengthen the usefulness of our proposal for the software-intensive industry.

8 CONCLUSIONS AND FUTURE WORK

This chapter presents an extended taxonomy for classifying GSE projects, which is based on a previous taxonomy [214], two empirical studies [24, 25] and on expert knowledge.

We addressed the first research question of this study (RQ1) by incorporating seven new dimensions into Smite et al.’s taxonomy, named “setting”, “software process type”, “software process distance”, “power distance”, “uncertainty avoidance”, “language distance” and “communication model”.

To validate the extended taxonomy and demonstrate its utility (RQ2), we benchmarked our proposal to two other taxonomies (Smite et al.’s taxonomy and Gumm’s taxonomy) and we also illustrate the usage of our taxonomy by classifying eight real finished GSE projects.

The results show that the extended taxonomy can help both researchers and practitioners by facilitating the reporting and understanding of GSE research. Although the new dimensions presented herein were identified in two studies related to effort estimation, the extended taxonomy presented in this chapter can be used to report any type of GSE study.

The list of dimensions in our extended taxonomy does not represent an exhaustive list of relevant GSE-related dimensions. Therefore, we intend to conduct further investigation to identify other dimensions that could be incorporated into the taxonomy, so that GSE projects could be classified in a more comprehensive way. More specifically, we intend to identify di-
Dimensions related to the way effort estimation processes are framed in GSE projects and how these factors relate to the effort of GSE projects.
1 INTRODUCTION

Large-scale software development is associated with high demands for continuous learning, which can be due to a number of factors such as i) the need for learning new competences in different functional areas of evolving software products; ii) professionals leaving the project (due to e.g. retirement, promotion or a more attractive job offer in another company). This means that companies often are demanded to recruit and train new people, eventually leading to the formation of new teams and onboarding of new developers. Understanding how to effectively and efficiently support developers and software development teams to learn what is required to do their work is one of the key challenges in software companies, especially when it involves ongoing product development.

Performance is said to improve over time, as individuals and teams accumulate experience and identify more effective and efficient ways to per-
form their work; this is called autonomous learning [60]. This means that there is a learning curve that relates experience and performance [4] (see Figure 3.1). Developers may learn individually or in teams. In the latter case, individuals may support each others’ learning through, for example, asking questions, seeking feedback, experimenting, reflecting on results, and discussing errors or unexpected outcomes of actions [64]. Performance of teams and individuals can also improve over time due to the participation in planned learning activities (e.g. training and coaching), which is called induced learning [60].

In this chapter, we report our experiences from establishing an approach to measuring how individuals and development teams learn over time (both autonomous and induced learning) and how their learning results relate to performance improvements. The reported experiences were collected in Ericsson in one product area that involves a number of collaborating sites. Although the approach gained a wider interest in the company, it has not yet yielded corporate-level practice.

Our investigation started with putting forward the following goals:

- Identify how learning relates to software development teams’ performance.
- Be able to measure a software development team’s potential to learn and consequently improve its performance over time.
- Be able to identify which development sites have larger learnability and performance potential.
- Be able to monitor a team’s learning results and performance over time.

We report herein the design of our investigation, specially focusing on which data was necessary to measure learning and performance improvements over time, how such data was retrieved, the preparation process it underwent, and the way it was analyzed. Finally, we summarize our experiences in the form of lessons learned.

The reminder of this chapter is organized as follows: Relevant concepts are provided in Section 2. Section 3 outlines the investigated case. Section 4 presents our measurement approach. Section 5 shows some example results of the measurement approach and the lessons learned. Section 6 provides some reflections on our experiences reported herein. Section 7 presents the validity threats associated with this chapter, followed by our conclusions and plans for future work in Section 8.
2 BACKGROUND

2.1 Performance

Performance in software development is expressed in different ways, like the productivity with which a particular work item is delivered or the number of post-delivery defects associated with a particular work item (quality). Performance is often linked with experience, which can be related to learning and the acquisition/accumulation of knowledge, skill and competence [18].

2.2 Learning

Learning is defined as “the acquisition of knowledge or skills through study, experience, or being taught” [166]. Learning can occur in three different ways [219]:

- **Formal learning** — Learning that occurs within an organized and structured context.
- **Non-formal learning** — Learning that occurs within planned activities that are not explicitly related to learning.
- **Informal learning** — Learning that occurs by carrying out daily life activities, which involves experiential or accidental learning.

Note that formal learning relates to induced learning, while non-formal and informal learning relate to autonomous learning.

2.3 Knowledge, skill and competence

Learning can lead to three main result types [219]:

- Knowledge is the result of an interaction between the capacity and the opportunity to learn. It is in general associated with formal learning, although can eventually be the result of non-formal or informal learning.
- Skill is the combination of mental and physical capabilities that demand practice to acquire. In many cases, the previous obtainment of knowledge is a pre-requisite for the achievement of a particular skill. It is in general associated with informal learning, although can eventually be the result of non-formal or formal learning.
- Competence is the extent to which individuals interact effectively with the environment. It describes personality aspects associated with better performance and higher motivation of individuals. It is acquired through informal learning.
The case and unit of analysis investigated herein is a large-scale distributed project in Ericsson that consists of the development of a telecommunication product originated in Sweden and has evolved for over 20 years. Many different technical and methodological changes were introduced over time, such as changing the programming language used to develop/maintain the product (from C++ to Java) and changing the software development methodology (from plan-driven to agile). Nowadays, the teams employ agile practices like iteration planning, coding standard and continuous integration.

The development and maintenance of this product involves over 150 employees working in teams located in Sweden, India, Italy, USA, Poland and Turkey. The offshore locations were added in response to the growing demands for resources and to implement market-specific customizations. The last expansion happened in Poland, where 2 teams were onboarded in the project in early 2016, following an expansion to India, where 10 teams were onboarded during the period between late 2014 and mid 2015. The site located in Turkey was decommissioned in 2014.

The work in the case follows agile software development principles. All teams are cross-functional by design, i.e. all members ought to be able to perform design, testing and programming duties. Teams receive an end-to-end responsibility for designing and implementing a work item, like addressing a customer’s product customization demand or fix trouble reports.

Software architects located in Sweden support the software development teams located in offshore locations (India, USA, Italy, Poland and previously Turkey). They support the remote teams by responding to questions related to the product’s software architecture and also by providing feedback through code reviews. In some urgent or particularly complex situations, the architects also participated in actual code implementation.

4 Measurement approach

As mentioned earlier, performance is often linked with experience, which can be related to learning and the accumulation of knowledge, skill and competence [18]. To analyze learning and performance in software development, it is necessary to collect the data associated with the work items performed by individuals or teams. Figure 3.2 shows an example of plot that relates accumulated experience (x-axis) and performance (y-axis).

These two variables in Figure 3.2 are expressed in our investigation as follows:
Performance – It can be expressed in terms of the productivity or lead-time associated with a work item (W). We propose to measure productivity ($P_{ki}$) with which team k carries out $W_i$ as the ratio between $W_i$’s complexity ($C_i$ – expressed in complexity points) and the actual effort ($E_i$ – expressed in hours) spent to carry out $W_i$ (Equation 3.1). The normalized lead-time ($NL_{ki}$) that team k takes to deliver $W_i$ is calculated as the ratio between $W_i$’s complexity ($C_i$ – in complexity points) and the lead-time ($L_i$ – in days) associated with $W_i$ (Equation 3.2).

$$P_{ki} = \frac{C_i}{E_i} \quad (3.1)$$

$$P_{ki} = \frac{L_i}{C_i} \quad (3.2)$$

Performance can also be expressed in terms of the quality of the work, but quality evaluation is a topic worth of a separate investigation, which we choose to omit from this chapter.

Accumulated experience – The accumulated experience ($A_{ki}$) of team k before starting $W_n$ is calculated as the summation of the
complexity associated with all the work items carried out by team \( k \) before \( W_n \) (Equation 3.3).

\[
A_{ki} = \sum_{i=1}^{n-1} C_i
\]  

(3.3)

Each point on the curve in Figure 3.2 represents a pair of a performance measure and \( A_{ki} \), i.e. each team participating in the investigation is to have its own two curves, one per performance measure. Evidently, data necessary for such calculations is rarely readily available. We have therefore added the data sources we have mined to draw the curves associated with the teams in the case (see Figure 3.3). These were:

- Time report spreadsheets – Documents that contain the actual effort spent by software development teams to carry out their work items (archival data). Used to measure performance in terms of productivity.
- Slot-plan spreadsheets – Documents that contain the planning information related to developers and teams assigned to particular work items. These spreadsheets help to identify the start and end dates of work items, and also the stability of the staffing in relation to a work item on a weekly basis, and team setup (archival data). Used to measure performance in terms of lead-time.
- Solution specifications – Documents that contain information on a work item design, including effort estimates (archival data). Used to support the measurement of complexity.
- Estimation sessions – Estimation sessions are conducted to calculate the complexity of work items. The complexity represents the difficulty to carry out the work. One work item is selected as a basis and experts are asked to measure the other work items in relation to it. They attribute a positive integer number to each work item (complexity points) using a planning poker approach.

4.1 Data preparation

To be able to analyze learning results and performance of individuals and software development teams over time, it is necessary to chronologically sort the work items carried out by a particular team. Furthermore, it is very important to identify whether the work items were carried out in parallel, to account correctly for all the experience accumulated until a particular moment in time.

Our approach involves a significant amount of data, which comes from different sources and has to be extracted by using different data collection methods. Thus, it is mandatory to perform a sanity check of the collected...
data. In doing so, it is possible to identify and correct inconsistencies that otherwise may lead to wrong results and conclusions.

Whenever an inconsistency or doubtful data observation is identified, we ask people involved with the data point (e.g. software architects and project managers) to clarify and correct the data observation (if necessary). In Section 6, we mention the inconsistencies we have identified so far and the way we have overcome them.

4.2 Data analysis

To analyze the data, in our approach we calculate descriptive statistics, charts for each team to show the relationship between performance and accumulated experience, and also conduct linear regression analysis to identify the strength of the relationship between experience (independent variable) and performance (dependent variable). Regression analysis is also useful to estimate the learning rates (i.e. the speed with which a developer or development team learns) and learning potential (the maximum performance that can be achieved by a developer or development team) of developers and development teams.

5 Examples and Lessons Learned

We faced many challenges and learned some lessons when establishing the aforementioned measurement, which we would like to share here. Thus,
in this section we illustrate our approach on sample measurements from three Indian teams that were onboarded in the case late 2014.

Productivity curves are one of the intended outcomes of our measurement process. Figures 3.4 to 3.6 show the productivity curve for each of the teams respectively, where the x-axis represents the accumulated experience (in complexity points) and the y-axis represents productivity (ratio between complexity points and the actual effort). It is important to emphasize that our intention here is not to compare the productivity or discuss the analysis of the curves. The curves will be used to identify two different trends: whether the teams are improving overtime and how stable is the knowledge acquisition process.

![Figure 3.4: Productivity curve of Team 1.](image)

To draw the curves illustrated in Figures 3.4 to 3.6, we identify the teams and their tasks first, and then look for team-task performance, which is not easy to carry out. First, we decided to focus on a sample of teams in one location. From the time report spreadsheets, we were able to extract the work items and the developers who have worked on these work items, and calculate the total work item actual effort and respective lead-times.

Most existing research on autonomous learning in teams in general and in software development teams in particular relies on quantitative data
analysis methods, such as regression analysis. However, to achieve statistically significant results, it is required to have a considerable amount of data points. Green [81] argues that a sample must be as big as 104 observations + K, where K means the number of independent variables in the model. Maxwell [143] extended the work by Green, accounting for the correlation between independent variables in the model. According to him, the smallest size of a sample that is able to lead to significant regression analysis results is 191 observations. These points in software teams are related to the work items performed by the teams. It was not possible to achieve this number of observations in our measurements yet.

Our data related to the recently on-boarded teams covers a two-year period, wherein the involved teams managed to implement only fifty unique work items. This appears to be way too few to enable statistically significant regression analysis. In practice, we believe that work items must have been broken down to smaller tasks carried out by individual team members or sub-groups of members. Unfortunately, we could not identify any sub-tasks in our case, which would have the actual effort data measured and involve cooperative work. Thus, the first two lessons learned are:

Lesson 1 – Gathering the required number of data points to perform regression analysis may be hard.

Lesson 2 – Consider what is a work item in a particular context and whether larger tasks can be broken down to smaller yet traceable and measurable sub-tasks.

Initially, we assumed that selecting teams from locations with more experience would enable the collection of more observations, since more mature teams would have completed many more tasks. However, to be able to analyze whether teams are learning by doing (autonomous learning), it is necessary to detect the inception date of the teams and the sequence
of each fulfilled work item \[4\]. Unfortunately, due to system and measurement changes, a lot of historical data in our case was lost or became difficult to trace. Therefore, the third lesson learned is:

**Lesson 3** – *Learning might be impossible to study if work item data from team inception is not available.*

Another challenge was to identify the sequence of each work item. In our case, many work items were carried out in parallel, which made it hard to identify the experience accumulated through on going work items. Hence, the fourth lesson learned is:

**Lesson 4** – *Detecting the step-wise accumulation of experience may be hard when teams carry out parallel work.*

Our next step was to estimate the complexity of each work item, which was achieved by consulting a group of software architects involved with the work items. During the first estimation session, we realized that some supporting material is needed in preparation for the meeting, to help the experts to recall the work items. However, the process of collecting the required material associated with old work items was not straightforward. Hence, our fifth lesson learned is:

**Lesson 5** – *Retrospective analysis of historical data may decrease data reliability or even be impossible to perform.*

Then, we connected work items with teams. To do so, we identified the team membership of each individual in the time report spreadsheets. A project manager helped us to gather this information. When doing so, we identified that it was very often the case that the composition of a team (or teams) involved in a work item did not match 100% the existing formal teams. This means that temporary task teams are sometimes composed to work on work items. A temporary task team is in general mainly composed by members of a particular formal team (determined by the formal membership) and complemented by members of other formal teams. We discussed this finding with the project responsible in Ericsson, to understand the reason why the work items were often assigned to temporary teams rather than to formal teams. The project responsible reported the following:

“The heavy ramp-up puts management in a tough spot. They cannot give work items to completely new teams to develop and at the same time keep the other teams intact. The new teams will get nothing done then. So they instead need to add new people to existing teams and move out some
experienced developers to build new teams instead. The “fork” approach is simply taken. This is from a practical point of view the best way to deal with the situation, but of course not optimal from a group learning perspective.

Management people in general and in particular in our site located in India has to very often deal with work items that risk becoming delayed. So, they deal with that by putting more experienced people on the prioritized task."

Thus, the allocation of work items to temporary teams was caused by tight deadlines and necessity to restructure the teams to facilitate on-boarding of new developers. Although this was also a common practice in other sites (e.g. in the Swedish site key developers are often relocated to urgent tasks), the magnitude of the staff changes in the studied offshore teams was much higher than expected. The sixth lesson learned is thus related to the notion of a team:

**Lesson 6** – Due to specific circumstances, work may be carried out by temporary teams rather than formal teams, which should be taken into account when collecting and analyzing the data.

This observation made us take a closer look at the task allocation and the staffing for each task. Iterations of data inspection and analysis were needed to finally have reliability in the data, which lead to a number of further challenges.

While looking into weekly slot-plans spreadsheets, we noticed that it is important to analyze further the developers’ engagement with the work items over time. For example, we identified cases in which a development team started a work item and afterwards handed it over to another team due to a sudden change in priorities. The time report spreadsheets in isolation do not allow for identify such a thing and, in this specific example, accounting only for this artifact would make us believe that the work item had been carried out by a large temporary team.

We have also noticed that even the formal teams can be unstable over time, since members can move from one formal team to another. This was identified by studying a number of team-related documents, received from one of the product managers responsible for the case.

All these challenges make more problematic the analysis of learning results and performance of formal teams, since in some cases teams change completely after the conclusion of work items and thus it is difficult to analyze the experience accumulated by a team. Hence, our seventh lesson learned is related to the case:
Lesson 7 – When teams are unstable or temporary teams carry out tasks, it might make more sense to focus on individual or holistic (site- or project-level) analysis of performance.

As suggested by Narayanan et al. [160], turnover of members in a team negatively correlates to team performance. In our case, the involved assets did not leave the company within the investigated time period, but the fact that developers work in temporary teams very often may have a negative impact on the performance of their formal teams. This could be one of the reasons why teams illustrated in Figures 3.4 to 3.6 did not demonstrate a stable increase in productivity performance.

When doing the sanity check for the data collected data, we identified some inconsistencies that could have led us to wrong conclusions. For example, while analyzing a particular work item, we found awkward the fact that the initial identified actual effort was too small for the complexity calculated by the software architects. So, we conducted a sanity check and identified that not all the effort spent in the work item had been reported; the work item was carried in 2015 and 2016, but only effort spent in 2015 had been reported initially. The sanity check allowed us to identify and correct this inconsistency. Therefore, the last lessons learned is:

Lesson 8 – Whenever data can be obtained from several different source and formats, it is fundamental to perform a sanity check to mitigate the chances that the data has inconsistencies.

6 Reflections and Discussion

Project staffing and team formation often become cornerstones in managing large-scale projects. Performance is expected to be higher when assigning tasks to the most competent people (e.g. in many situations software architects). However, while cultivating individual competence in focused areas might be economically feasible short term, research suggests that higher performance can be achieved by cultivating teamwork and thus fostering group learning [64], [65]. Companies accumulating what is regarded as social capital, i.e. knowledge resources that can be obtained through teamwork and networking, are rewarded with higher performance [222]. To achieve such a thing, it is recommended to keep team members working closely with the other members of their formal team, reducing as much as possible the fulfillment of work items by temporary teams with high team member turnover.

The differences between formal teams and temporary teams might have another important implication. It is fair to assume that formal teams will
show higher performance than temporary teams. However, the vast major-
ity of performance-oriented studies in software engineering are based on
archival data, often coming from publicly available repositories. However,
what has been studied as a “software team” is not always clear and/or
accurate (formal or temporary). As such, many team and learning related
aspects in these studies might be overlooked or impossible to judge due to
the missing or incomplete data.

Hackman has defined a team as a work group that exists within the con-
text of a larger organization and shares responsibility for a team product or
service [84]. Katzenbach and Smith suggest looking into four key elements
that determine a team – common commitment and purpose, performance
goals, complementary skills, and mutual accountability [113]. Finally, there
are project teams of temporary nature. The question is then whether it is
sufficient to simply put individual developers together and expect them to
work effectively. Our experience so far suggests that the answer is no.
Thus, we urge software engineering researchers to clarify whether what
is being studied are formal or temporary teams. It is also important to
distinguish between what is being learned by individuals within a team
(individual learning) and what is learned by a team as an “atomic” entity
(group learning), since group learning is different than the aggregation of
what is learned by team members; rather, team learning is about a new
understanding shared by all the team members about how the team is
expected to function [80].

7 VALIDITY THREATS

The lessons learned reported herein have the following main validity
threats:

- **Reliability** validity threats are related to the repeatability of a study,
i.e. how dependent are the research results on the researchers who
conducted it [181]. The lessons learned reported herein are the result
of the work of two different researchers. Furthermore, they were ver-
ified with the company representatives to avoid false interpretations.

- **Internal** validity threats are related to factors that the researcher is
unaware of or cannot control the extent of their effect in the investi-
gated causal relationship [181]. The experiences reported herein in-
volved only part of the data we collected (mainly data from India).
Thus, the lessons learned could be different if we had analyzed all
the available data, which involves sites from 6 different countries.
However, we mitigated this threat by discussing our findings with
representatives of Ericsson.

- **External** validity threats limit the generalization of the findings of the
investigation [181]. The approach and the lessons learned reported
herein are associated with the context of our research in Ericsson. However, we believe that several software development companies have similar context and thus can benefit of the contributions of this chapter.

8 CONCLUSIONS AND FUTURE WORK

In this chapter, we reported our experiences from establishing an approach to measure learning and performance of developers and software development teams in Ericsson.

While carrying out our analysis, we encountered multiple challenges related to archival data collection and analysis. We believe that these challenges are not unique and other companies can learn from our experiences. Thus, we reflected all the challenges we faced in the form of lessons learned. Evidently, our premise is that the measurement is done on the already available data. More recommendations can be derived from our lessons learned for companies that plan to perform similar measurements and are able to influence the data collection practices.

The content of this chapter is a part of a wider ongoing investigation in Ericsson that aims at supporting the learning process and performance improvements of developers and software development teams in large-scale distributed software projects. Therefore, we intend to continue the investigation, accounting for the lessons learned and using the measurement approach reported herein.
SOFTWARE ARCHITECTS IN LARGE-SCALE DISTRIBUTED PROJECTS: AN ERICSSON CASE

1 INTRODUCTION

Much has been written about software architects, their roles [77, 97, 123, 178, 187], their responsibilities [20, 124, 194] and how they are “made” [52]. Martin Fowler proposed the following classification of software architects [77]:

- *Architectus Reloadus* makes important decisions early on, ensuring a system’s conceptual integrity.
- *Architectus Oryzus* addresses problems in a project by closely collaborating with the developers.

Philippe Kruchten summarized software architects’ responsibilities, including defining the architecture; maintaining its architectural integrity; and consulting with, helping, and coaching development teams [123]. Paul Clements and his colleagues described architects’ skills and knowledge, such as communication and learning skills [45]. Grady Booch [22] and Frank Buschmann [35] have written on aspects of architects’ daily work.

However, most of the vast literature on software architects has focused on the architect as an “one-soldier army”. This seems reasonable in small projects but infeasible in large-scale projects, which require teams of architects [123]. Because many large-scale projects are spread across multiple locations, we raise two research questions:

- **RQ1** – What are software architects’ roles and responsibilities, and how are they organized in large-scale distributed software projects?
- **RQ2** – How does the addition of new teams in remote locations influence the architects’ role?
  - **RQ2.1** – What is the effort needed to guard and mentor?
  - **RQ2.2** – What are the problems prevented by software architects?

Here, we share our results from studying the development of a large, distributed software product at Ericsson, a global leader in telecommu-
nations with sites worldwide. We focus on the architects steering this product’s development.

The remainder of this chapter is organized as follows: Section 2 presents the research design. Section 3 presents the results of this chapter, which are discussed in Section 4. Section 5 presents the associated validity threats. Finally, a summary with our conclusions and view on future work is provided in Section 6.

2 Research Design

We conducted a case study [181] to investigate the development of a telecom product that has evolved for more than 20 years. At the time of our investigation, the product employed more than 150 employees (including 15 product-level architects, 20 design leads, 100 developers working in 22 teams, and 15 other supporting roles) in Sweden, India, Italy, the US, and Turkey. One team comprised only product-level architects; each of the other teams consisted of a design lead and two to eight developers.

The results presented here are based on the data from 240 tasks carried out from January 2014 to June 2016. We employed the following data collection methods.

First, we performed archival research. We analyzed managerial documents (plans and reports) related to 26 product customization tasks (PCs), measuring the effort architects expended for development and support. Twelve teams participated in these PCs (for more on these teams, see the main article). We also surveyed the Gerrit code review repository to determine the number of defects identified through code reviews completed for a random selection of tasks: 10 PCs, 188 trouble reports (TRs), 9 product improvements (PIs), and 11 standardizations of market features (MFs). We manually categorized the defects using the ISO/IEC 25010:2011 standard [102]. The product’s unit manager validated the collected archival data.

Next, we conducted unstructured interviews with the unit manager and product-level architects. We collected their knowledge about how the architects were organized, the architects’ roles and responsibilities, and the development teams’ maturity.

Finally, we held four workshops with the unit manager and product-level architects to measure the 26 PCs’ task complexity – the difficulty of performing them. We selected one of the PCs as the baseline and had participants measure the other PCs in relation to it. Using planning poker, they attributed a positive integer (complexity points) to each PC.

To check the assigned measures, the architects consulted technical-solution specifications whenever needed. The PCs, PIs, and MFs resembled independent projects because they had start and end dates, a responsible team,
and expected results. In general, PCs take 1 to 3 months, PIs take 1 to 2 months, and MFs take 2 to 6 months.

3 RESULTS

In this section we present the results that answer the two research questions of this chapter.

3.1 RQ1 – Roles, responsibilities and organization

The architectural paradigm employed in a software product’s architecture affects how software architects carry out their work [131]. Whereas most new products at Ericsson have a modular or service-oriented architecture, legacy products, such as the case we describe here, have a monolithic architecture.

The scale and complexity of Ericsson’s software products, together with the distribution of development teams across multiple locations, also impact how architects perform their work. When teams and development locations are added to a project, it requires extra support to guard product integrity and mentor the teams while they climb the learning curve.

In distributed environments, this introduces several challenges. One Ericsson software architect referred to communication overhead:

“it would be much easier to explain to the developers things about the product’s architecture if they were in Sweden, because most of the time we only communicate via the code review tool and email”.

Another architect pointed out the necessity of guarding and control:

“When multiple teams work together on a task, the developers across teams do not communicate as well as within a team. Therefore, they often introduce task conflicts that we nd in their code, which otherwise the teams would not identify themselves”.

To ensure cross-team and cross-location alignment and to guard the complex system architectures, Ericsson architects are organized in three levels, as shown in Figure 4.1.

At the system level, a team of architects of the type Architectus Reloadus ensures that the products included in a system can communicate with each other. They closely collaborate with product-level architects and coordinate the design of the features, interfaces, and protocols used in the system – that is, the system architecture. They also review product-level solutions and guard the system’s evolvability. These architects are mostly in Sweden.

95
At the *product level*, a team of architects fulfills the *Architectus Reloadus* and *Architectus Oryzus* roles. These architects, who are experienced, skillful developers, lead the design of technical solutions and guard the integrity and evolvability of the product and the system architecture. They collaborate closely with system-level architects and software development teams throughout solution implementation. They steer software design, ensure that design rules are followed, promote the use of design patterns, and ensure code quality through code reviews. They help guard the architecture and the deliverables’ quality. They also mentor the software teams by sharing their knowledge, providing feedback through code reviews, suggesting improvements, and helping out in urgent situations to speed up development. These architects are also mostly in Sweden, and they’re this article’s main focus.

At the *development team level*, senior developers called *design leads* (or team-level architects) support the design of technical solutions. They ensure that their team follows the design rules and design patterns, which generally are defined by product-level architects. They also help product-level architects guard a product’s architectural integrity and evolvability, and help ensure the quality of deliverables through code and documentation reviews. The more expertise and experience design leads have, the more autonomous their teams are; they can provide more support to product-level architects with mentoring and guarding.

The three types of architects interact in the following way:

1. A prioritized business use case is assigned to a system architect.
2. The system architect performs impact analysis and forms a virtual team with representatives from the impacted products to define the solution.
3. The product architect specifies the product-level design with the design lead from the employed development team (team members are included in the design work when applicable).

4. The development team performs the detailed design and implementation with the product architects’ support.

The product and development team levels might vary slightly among different Ericsson products. In our case, the product level was “thicker” and centralized almost all responsibility for the overall design of the technical solutions to be implemented in the product. The overall responsibility for guarding the product architecture’s integrity and evolvability was also centralized. This means that even the more experienced teams were never fully autonomous. More recent products with a service-oriented architecture have a much thinner product level. That is, they have approximately 50 percent fewer architects, and the teams shoulder much of the responsibility.

3.2 RQ2 – *The influence of team maturity, scale and distribution on the architects’ role*

In our case study, the main factor affecting the extent of the product-level architects’ involvement was the development teams’ maturity. To visualize the product-level architects’ amount of work and area of responsibility, we designed the authority matrix (see Figure 4.2), which was inspired by the work of Hackman on teams [85, 86]. The curve in the matrix shows that as a team matures, it should get more independent and require less support from the product-level architects. The four activities on the y-axis are those that concerned the product-level architects the most. The x-axis indicates the four levels of team maturity.

At level A, an immature team has no or very basic knowledge about the product’s code and architecture. Level-A teams require much mentoring and support from the architects, even when implementing (coding, testing, and documenting) non-complex technical solutions. Product-level architects guide the teams during the implementation and are actively involved, with both reviewing and approving code.

At level B, teams have sufficient knowledge to implement non-complex technical solutions without much guidance. More knowledgeable team members and design leads review the implemented code, but product-level architects still review most of the design and code.

At level C, experienced, mature teams with good knowledge of the product architecture implement complex technical solutions that have a significant architectural impact. These teams review and approve their own code. Product-level architects are involved only in approval when the architec-
ture’s critical components are affected. Level-C teams also perform more solution design. Product-level architects support these teams by mentoring the design of technical solutions and providing on-demand guidance in the initial implementation stages.

At level D, very experienced and rather autonomous teams implement complex technical solutions independently, even ones affecting the product architecture’s critical components. The code they implement does not need product-level architects’ approval. Level-D teams also drive the design and review of technical solutions. Product-level architects are involved only in the technical-solution design.

Our authority matrix is based on the product-level architects’ experience and the data we collected and analyzed in our case study (see Section 2). In particular, our statistical analysis of the performance data confirmed that the more immature teams required more support from the product-level architects.

Figure 4.3 shows the teams’ distribution. The Turkish team was classified as level A. The five recently onboarded Indian teams started at level A; four of them progressed to level B during our study. The four Swedish teams and the US team were classified as level C. One Swedish team temporarily reached level D (product-level architects joined the team), and a temporary level-D team comprised product-level architects, developers, and design leads from Sweden and India.

---

1 A critical component contains core functionality that executes key operations.
RQ2.1 – The effort needed to guard and mentor

The teams’ maturity also determined the number of required product-level architects. In our case study, the rule of thumb seemed to be that each level-C team needed approximately one-half of a product-level architect and that each level-A or level-B team needed one product-level architect. As we mentioned before, no permanent level-D teams were in our case study. In cases involving critical work demanding specific architectural knowledge and quick development, product architects might temporarily join a team to engage in coding and testing. Such teams can qualify as level-D teams.

Figure 4.4 shows the effort that product-level architects spent supporting 12 teams (the teams we mentioned earlier, except the temporary Swedish team) performing 26 product customization tasks. The less mature teams (levels A and B) received significant support from the product-level architects. In contrast, mature teams (levels C and D) worked more autonomously. We verified these results’ statistical significance using the Kruskal-Wallis one-way analysis of variance (significance level = 0.01, p-value = 0.008493).

Part of the effort involved mentoring, which the product-level architects provided through meetings (videoconferences or teleconferences with remote teams) or synchronous chat. Most of the effort involved code reviews, which also provided mentoring opportunities and involved clarification of architectural principles and improvement recommendations. In some
cases, support included detailed instructions that even contained code samples to facilitate the understanding of the design.

**RQ2.2 – The problems that product-Level architects prevented**

To ensure the product architecture’s integrity and evolvability, product-level architects review new and modified code to identify defects that might harm the product architecture, induce technical debt, and complicate the architecture’s evolvability. Although Ericsson devotes much attention to rigorous regression testing, not all defects can be caught automatically, especially non-functional maintainability defects [70]. Maintainability defects can lead to code entropy [61], affecting the architecture’s integrity.

At Ericsson, new developers working on legacy products often have problems understanding the code’s structure, which leads to the fear of changing existing code and the tendency to duplicate code [87]. This behavior is common in immature teams, and architects play a vital role in detecting such defects through code reviews, making code review a crucial asset.

Figure 4.5 contains results about defects revealed through code reviews performed on 220 tasks. We focused on two subcategories of maintainabil-
ity defects closely related to Ericsson architects’ guardian role: modifiability and analyzability [102].

Figure 4.5: Amount of defects revealed by product-level architects and teams.

The code reviews revealed 1,727 defects; product-level architects discovered 72.38% of those defects, and the teams discovered 27.62%. Of those defects, 85.18% were maintainability defects, and the product-level architects revealed 72.40% of them. Considering that testing techniques do not easily reveal maintainability defects, the product-level architects’ active participation helped in early detection of defects that otherwise could have affected the product architecture’s integrity and evolvability.

4 DISCUSSION

Although the software architects’ hierarchy, roles and responsibilities we described are not unusual [8, 123], they are not trendy, either. So, in this case study, why did not Ericsson fully embark upon trendier approaches such as microservices [7] and agile ways of working?

The answer is: because of legacy code and distributed development. Although this case study implemented many agile practices such as continuous integration, continuous delivery and DevOps, certain limitations prevented fully exploiting the new approaches.

The product’s monolithic architecture made relying on coordination by mutual adjustment problematic [147]. However, this does not automatically mean that Ericsson applied traditional coordination and control. The architects’ hierarchy was not traditional in the sense that people higher up in the hierarchy could overrule the ones further down. Rather, it was a network of architects with different focus areas who were involved in decisions related to their competences. On the product level, the architects and other experienced developers had designated approval rights based on their competences. They governed this approval structure and process as they saw fit.
This governance structure was similar to large distributed open source projects such as Eclipse and Android. That is, the open source community has also concluded that large development projects sometimes require a centralized approach to secure the software’s quality.

The architects’ hierarchy was also an answer to the challenges of distribution. Ericsson is no exception to the trend of offshoring parts of development to low-cost regions; much of its product development has become global. However, many offshore sites are infamous for high attrition rates. This means that employee turnover results in continuous loss of expertise and the neverending demand for mentoring and steering. Knowledge loss also occurs from frequently moving resources in and out of products because of rapidly changing business needs. In such situations, the centralization of architectural decisions such as we described here secures the architectural knowledge’s stability and its availability for new developers and teams.

The speed with which new technologies are introduced has compelled Ericsson’s software architects to adapt quickly. For example, cloud computing forces architects to rethink existing architectures and use different approaches to design new ones. It is no longer only about designing software to conform to a particular hardware specification, because virtual environments hide the underlying infrastructure.

At the same time, many new technologies and methods have reduced Ericsson architects’ workload. For example, continuous integration and continuous delivery tools require less manual effort to ensure that code changes do not harm a product architecture’s behavior. To implement continuous integration and continuous delivery in our case study, they adapted the product’s architecture to maintain full backward compatibility despite the frequent releases – for example, through strict version control of interfaces.

Ericsson also employs DevOps. In our case study, two product-level architects were dedicated to ensuring smooth collaboration between Ericsson’s development team and one customer’s operation team. This shortened feedback loops, which led to better interaction with Ericsson’s customers.

Another trend influencing the role of Ericsson architects is the agile transformation. To be able to benefit from agile ways of working, teams are expected to have more autonomy, and architects serve primarily as mentors supporting the teams. Although Ericsson has tried to increase team autonomy for the case study’s product, the company has focused on increasing autonomy for more recent products not limited by legacy architectures.

Finally, Ericsson architects must be prepared to work on products developed at multiple locations. As in many other large organizations, corporate cultures and ways of working differ among Ericsson’s distributed
units. So, software architects are often expected to possess strong leadership skills and the ability to align the philosophy, traditions, and attitudes toward quality and maintainability across all development locations.

5 Validity Threats

The threats to validity related to this chapter are discussed using the categories reliability, internal, construct and external validity described by Runeson and Höst [181]. Due to the use of statistical inference, we also discuss conclusion validity [198].

Reliability

Reliability is related to the repeatability of a study, i.e. how dependent are the data and analysis on the involved researchers [181].

This threat was minimized by involving all three authors in the design and execution of the investigation. We also developed an explicit case study protocol [181]. Furthermore, the observations were verified by Ericsson’s representatives to avoid false interpretations and inconsistencies. Nevertheless, part of the data was collected through interviews and is highly dependent on the involved interviewees. This means that it might be very difficult to obtain the same values with other informants.

Internal validity

Internal validity is related to factors that researchers are unaware of or cannot control regarding their effect on the variables under investigation [181].

To mitigate the associated threats, we made an attempt to account for as many confound factors as possible through interviewing people with different roles and by using existing literature (data and methodological triangulation). To mitigate investigator bias, the three authors were involved in the interpretation of the results (investigator triangulation). To mitigate interviewee bias, we interviewed people with different roles (data triangulation).

Construct validity

Construct validity reflects how well the measures used actually represent the constructs the study intends to measure [181]. The main threat to construct validity related to this chapter is that we used only one method to measure each construct.

To mitigate this threat, we collected data from different sources using different methods (data and methodological triangulations) to strengthen
the produced evidence. Moreover, we verified the collected data with Ericsson’s representatives.

*External validity*

External validity is concerned with the generalization of the findings [181]. Results of case studies are strongly bound by the context of the selected cases. This investigation focused only on one product from one company, which further limits the generalization of the findings. Nevertheless, the findings of this chapter may be of interest for companies carrying out large-scale globally distributed projects with large amounts of complex legacy code. To allow transferability of the findings, the context of this investigation was comprehensively detailed.

*Conclusion validity*

Conclusion validity is concerned with the correctness of conclusions regarding relationships in the analyzed data [198]. The main threats to conclusion validity are the low reliability of measures (the amount of noise related to a measure) and low statistical power. To mitigate low reliability, the collected data was verified with Ericsson’s representatives. In relation to statistical power, we were limited to the available data, which covered only two years of tasks carried out in the case.

6 CONCLUSIONS AND FUTURE WORK

In this chapter, we presented the results of an investigation that focused on identifying the software architects’ roles and responsibilities, and how they are organized in large-scale distributed software projects (RQ1). Furthermore, we investigated the influence of team maturity, scale and distribution on the architects’ role (RQ2).

Regarding RQ1, we identified that to deal with scale, distribution and maturity, the architects are organized in a semi-hierarchical manner (system-level, product-level and team-level architects), with the product-level architects having the main responsibility for guarding the product architecture and mentor new software development teams. Scale, distribution and legacy demanded a more centralized approach to avoid architectural entropy.

In relation to RQ2 (and subquestions), the results suggest that team maturity is the factor that relates the most with supporting effort; the bigger the team maturity, the lower the associated supporting effort. We also learned that product-level architects were responsible for identifying a wide variety of defects that, if not revealed, could have specially impacted the evolvability of the product’s architecture.
The case study reported herein just covered two years of data (mainly product customization tasks). Thus, we plan to continue the data collection process to be able to strengthen the empirical evidence produced in this investigation. Furthermore, the authority matrix proposed in this chapter can be used to track the evolution of teams in terms of team maturity. Hence, we plan to investigate how teams onboarded in this case have evolved over time in terms of team maturity, as well as team performance (e.g., productivity, autonomy and code quality).
LEARNING AND PERFORMANCE EVOLUTION OF IMMATURE REMOTE TEAMS IN LARGE-SCALE SOFTWARE PROJECTS: AN INDUSTRIAL CASE STUDY

1 INTRODUCTION

Companies worldwide are developing software in a globally distributed manner (Global Software Engineering – GSE) to achieve benefits such as reduced time-to-market and access to skilled people all over the world [21, 49, 88, 172]. However, geographical, temporal and cultural distances make coordination and communication more challenging in GSE environments. GSE projects often involve a large number of people (large-scale projects) and the scale amplifies the distribution-related challenges [57]. The combination of scale and global distribution may lead to problems, such as more software defects [68], and schedule and budget overruns [90].

In addition to the challenges imposed by scale and global distribution, GSE projects are often associated with the development of products with long life cycles [67, 206]. Long life cycles are associated with large amounts of (often complex) legacy code [206] and the onboarding of many different developers. Onboarding of new developers may occur for different reasons: to replace retired developers; to replace developers that left the company for another job; or to increase the existing workforce. Developers may also be relocated within the company and, thus, onboarded to perform different types of tasks that require new competence build up. While developers may be onboarded at multiple occasions during the life cycle of a large-scale GSE project, the onboarding processes may be more challenging in this context, due to the aforementioned scale and distribution challenges, and the difficulty to learn and change legacy code. [19, 43].

Learning is one of the main components of onboarding [136]. Existing research shows that learning is related to performance of developer-
s/teams; the more product, technical and methodological knowledge a developer/team\(^2\) has, the better a developer/team’s performance \([18, 68, 99]\). Learning is also dependent on the nature of the task to learn and the knowledge to acquire. When the knowledge is abstract, which is the case in software development, the novices may have difficulty to learn and evolve their performance, even in the presence of expert trainers/mentors \([93]\).

While novice/immature\(^3\) developers/teams have a hard time to learn and evolve (in terms of performance) in large-scale GSE projects with long life cycles, remote immature developers/teams may have an even harder time. This is mainly due to the fact that distribution amplifies the challenges of acquiring the required knowledge \([89, 122, 172, 210, 213]\). A better understanding about learning and performance evolution of remote teams may help organizations to be more successful when onboarding software developers in GSE projects.

Although the relationship between learning and performance evolution has been investigated by software engineering researchers \([18, 100, 199]\), there is limited knowledge about how scale, distribution, and complex legacy code impact learning and performance in globally distributed projects.

In this chapter, we intend to explore performance of newly onboarded offshore teams through a longitudinal analysis of performance data collected through a case study conducted in Ericsson\(^4\), a large company that offers different telecommunication related products. The case is a large-scale software product that has been developed over the last 20 years. We relate our findings to existing work on the link between performance differences in mature and immature teams and the challenges in learning. Furthermore, we explore performance evolution of immature teams onboarded on a distance in a large-scale project involving complex legacy code. The research questions that drove our study are as follow:

- **RQ1** – How does the performance of immature remote development teams evolve over time?
- **RQ2** – How does the performance of immature remote development teams differ from mature development teams?

The contribution of this chapter is two-fold:

- A quantitative and qualitative analysis of the learning processes and performance evolution of immature remote development teams in a large-scale industrial case involving complex legacy code.

---

\(^2\) We consider a team as “a work group that exists within the context of a larger organization and shares responsibility for a team product or service” \([84]\).

\(^3\) By immature developer, we mean a developer that has no or very little knowledge about a legacy product under development/maintenance, and knowledge about the associated technology stack.

\(^4\) www.ericsson.com
• Recommendations to support the learning processes of immature remote development teams.

The remainder of this chapter is organized as follows: Section 2 describes the background and related work. Section 3 presents the research design. Section 4 presents the results. A discussion, which includes the implications of our work for academia and industry, is presented in Section 5. Validity threats are discussed in Section 6. Finally, a summary with our conclusions and view on future work is provided in Section 7.

2 BACKGROUND AND RELATED WORK

Research on team learning and performance originated from the social sciences and is within the larger field of organizational learning. Organizational learning and knowledge management is a very wide field characterized by a multitude of disciplinary perspectives [5]. Team learning and performance research has gained a large attention from various disciplines varying from the shop floor to the executive suite [48]. This stream of research brought a number of new topics also into the repertoire of software engineering and information systems research.

In this paper, we focus on one particular type of learning and its links to performance, i.e. we study the impact of autonomous learning on performance evolution over time, as novice developers gain knowledge and experience. In the following, we present some background about learning, followed by related work on performance studies in software engineering and in globally distributed projects.

2.1 Learning and performance

Learning is defined as “the acquisition of knowledge or skills through study, experience, or being taught” [166]. To better understand the relationship between learning and performance, we hereby describe the key concepts related to learning [219]:

• **Knowledge** is the result of an interaction between the capacity and the opportunity to learn. It is in general associated with formal learning (organized and structured learning), although can eventually be the result of non-formal (unplanned learning) or informal learning (experiential or accidental learning).

• **Skill** is the combination of mental and physical capabilities that demand practice to acquire. In many cases, obtaining knowledge is a pre-requisite for achieving a particular skill. It is in general associated with informal learning, although can eventually be the result of non-formal or formal learning.
• **Competence** is the extent to which individuals interact effectively with the environment. It describes personality aspects associated with better performance and higher motivation of individuals. It is acquired through informal learning.

Two types of learning can occur within teams:

• **Individual learning**, where individuals learn from other members of a team [80].

• **Group learning**, a process of reflection and action, which encompasses different learning behaviors, such as asking questions, seeking feedback, experimenting, reflecting on results, and discussing errors or unexpected outcomes of actions [64]. Group learning leads to a shared understanding at group level and a change in a team’s repertoire of potential behaviors [218].

Performance is often linked to experience, which can be related to learning and the acquisition and accumulation of knowledge, skill and competence [18]. Performance is also said to improve over time, as individuals and teams accumulate experience and identify more effective and efficient ways to perform their work, through autonomous learning or learning by doing [60].

An important concept associated with autonomous learning and performance evolution the learning curve phenomenon [4]. Performance of teams and individuals can also improve over time due to the participation in planned learning activities (e.g., training), which is called induced learning [60].

A learning curve describes the performance of teams or individuals in a mathematical way. It was proposed by Wright [223] based on observations of how the costs associated with assembling airplanes decreased as the involved workers accumulated experience in doing the same type of task (autonomous learning).

Learning curves can be modeled using univariate or multivariate models, e.g., log-linear, hyperbolic and exponential models [4]. Log-linear models are most frequently employed due to their simplicity. The original model proposed by Wright [223] is a log-linear model, which is represented by Equation 5.1.

\[
Y = CX^b
\]  

(5.1)

In Equation 1.1, \(Y\) is the average time (or cost) per unit demanded to produce \(X\) units (cumulative experience) and \(C\) is the time (or cost) to produce the first unit. The parameter \(b\), also known as learning rate, represents the slope of the learning curve. Equation 5.1 has been modified by the research community, resulting in other versions of the aforementioned power law, e.g., the De Jong and S-curve models [4].
2.2 Performance and learning in software engineering

In the following, we first outline performance studies in the context of autonomous learning in software teams and then summarize the research about the performance of remote teams onboarded in ongoing software projects. The key studies linking performance with different impact factors are summarized in Figure 5.1 and Table 5.1. Finally, we outline the research work conducted in the area of global software engineering and onboarding of offshore developers.

Figure 5.1: The impact of different factors on performance and related studies.

A number of researchers studied the relationship between autonomous learning/accumulation of experience and performance (learning curve phenomenon) [18, 99, 100, 114, 160, 199, 226]. Huntley [100] explored the relationship between autonomous learning and debugging cycle times in open source projects. His results suggest that the accumulation of experience impacts debugging cycle times more in mature projects than in emerging projects, i.e. the learning curve phenomenon is dependent on the context of each project. Tüzün and Tekinerdogan [199] investigated the impact of the learning curve phenomenon on the return of investment (ROI) in software product line engineering and concluded that the learning
Table 5.1: Impact factors’ details.

<table>
<thead>
<tr>
<th>Impact factor</th>
<th>Reference</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 – Specialized experience</td>
<td>[18, 99, 100, 114, 160, 199, 226]</td>
<td>+</td>
</tr>
<tr>
<td>F2 – Diverse experience</td>
<td>[18, 160]</td>
<td>+/-</td>
</tr>
<tr>
<td>F3 – Project maturity</td>
<td>[114]</td>
<td>+</td>
</tr>
<tr>
<td>F4 – Team stability</td>
<td>[160]</td>
<td>+</td>
</tr>
<tr>
<td>F5 – Team familiarity</td>
<td>[99]</td>
<td>+</td>
</tr>
<tr>
<td>F6 – Scale</td>
<td>[184]</td>
<td>-</td>
</tr>
<tr>
<td>F7 – Team coordination</td>
<td>[58]</td>
<td>+</td>
</tr>
<tr>
<td>F8 – Goal-orientation</td>
<td>[58]</td>
<td>+</td>
</tr>
<tr>
<td>F9 – Team cohesion</td>
<td>[58]</td>
<td>+</td>
</tr>
<tr>
<td>F10 – Shared mental models</td>
<td>[58]</td>
<td>+</td>
</tr>
<tr>
<td>F11 – Global distribution</td>
<td>[17, 62, 122, 149, 213, 215]</td>
<td>-</td>
</tr>
</tbody>
</table>

curve has a clear impact on the ROI of software development companies, although such an impact gets lower when the number of products of a particular software product line increases. Zorgios et al. [226] proposed an explanatory theory for team learning related to software development. The authors modeled the interaction between learning rates of development teams and improvements in their productivity, establishing a causal relationship between the human capital and different types of learning of organizational teams and the productivity curve.

Autonomous learning/accumulation of experience through specialized tasks or particular roles has been associated with increased individual productivity [18] and quality [99, 114, 160]. Huckman et al. [99] investigated the relationship between role experience, software quality and budget/schedule adherence in 1,004 development projects, and found that the more role experience, the lower the number of post-delivery defects and the greater the adherence. Kim and Jiang [114] analyzed the impact of accumulation of experience on bug ratio per developer in open source projects. The results suggest that direct bug-fixing experience decreased developers’ bug ratio [114]. Boh et al. [18] investigated the learning curve phenomenon on the individual, group and organizational levels using 14 years of data regarding a project. Narayanan et al. [160] focused on the rela-
tionship between task variety and individual learning in software maintenance tasks using six years of data regarding a project. The results of their studies suggest that the more diverse experience the higher the productivity [18, 160]. Furthermore, Boh et al. identified that diverse experience has a larger impact on the productivity associated with group tasks [18] than homogeneous experience. However, Narayanan et al. identified that too much variety can also hinder team productivity [160].

Researchers have also studied what influences performance. Brown and Eisenhard [32] presented a comprehensive relationship model of success factors in product development that includes the impact and mediating effects of team composition factors, team organization of work and group processes, project leadership, senior management, customer and supplier involvement, and product concept effectiveness.

One vein of research of particular interest for our work focuses on specific aspects of group learning and their influence on the learning outcomes and learning behaviors. In particular, a number of studies have quantified the impact of team-related factors on group learning and performance (team size, team familiarity, team stability, team coordination, goal-orientation, team cohesion, shared mental models) [99, 160, 184]. Huckman et al. [99] investigated the relationship between team familiarity, software quality and budget/schedule adherence, and found that the higher team familiarity, the lower the number of post-delivery defects and the greater budget/schedule adherence. Narayanan et al. [160] investigated the impact of turnover (team stability) on performance and found that the higher the team turnover, the lower the productivity. Scholtes et al. [184] focused on identifying the relationship between team size (scale) and productivity in open source projects. Their results suggest that there is a negative relation between team size and productivity, which is in line with Brooks’ law [31] and the Ringelmann effect [174]. Finally, Dingsøyr et al.’s [58] systematic literature review on team performance software development shows that team coordination, goal-orientation, team cohesion, shared mental models and learning strongly improve team performance.

2.3 Performance and learning in global software engineering

Current research on performance in global software engineering supports two types of decisions: whether to offshore or not, and how to organize distributed projects. To support these decisions, researchers have compared productivity and/or quality performance of collocated and distributed teams, as summarized by Nguyen-Duc et al. [162]. The results of their study suggest that geographical dispersion has a negative impact on team productivity and software quality, while temporal dispersion has been linked to decreased quality only. However, most of the primary studies
in this area seem to be devoted to greenfield projects and take a static standpoint. There are very few performance studies that distinguish between projects distributed from the beginning and projects that onboard remote teams at later stages.

The research on learning and performance evolution of remote teams onboarded in ongoing projects is especially scarce. Time to climb up the learning curve and acquire the required knowledge to perform as expected is of particular interest for companies that make decisions to onboard remote locations. Mockus and Weiss [149] studied individual developers working on non-trivial modification requests and found that remote developers may reach full productivity in approximately 15 months, after three months of project training prior to the actual work. A number of perception-based studies and experience reports suggest that the learning process may take from 12 months [62], up to three [215] and five years [122, 213] or even longer [17].

Industrial experiences shared by Carmel and Tija [40] demonstrate that efficiency of new developers after a transfer can decrease down to 20% of the relative efficiency of the original unit and rarely achieve full recovery. Smite and van Solingen [211] have found that it took longer for remote offshore developers to climb up the learning curve in comparison with the developers onboarded onsite due to the distance to the sources of requirements and development knowledge, and the lack of local developers with experience. They also found that onsite developers received more attention and training, while offshore developers were expected to be mentored and trained by the outsourcing vendor company. In our previous work [27], we studied the importance of mentoring and coaching for onboarding offshore teams and found that architect support is indispensable for the new teams onboarded on a distance in legacy projects with the high amount of complex legacy code, and supports autonomous learning in a number of ways, including extensive feedback through code reviews and help in finding the solution to particularly complex problems.

Another vein of research relevant to our work is the particular case of onboarding of remote developers who later receive the full responsibility for the work called software transfers. Offshore software transfers have been found to result in an overall performance decrease in terms of release frequency [105], lead-times [106] and quality (number of defects) [105, 106]. Understanding the capabilities of the new teams and monitoring the learning curves in such contexts is particularly important for organizations to be able to make an informed decision about the ability of the new developers to take over the responsibility for the transferred product. This is also the context of the case studied in this paper.

To summarize the related work, it is fair to say that the majority of studies related to learning and performance in global software projects has focused on comparing distributed teams to collocated teams, while the
number of longitudinal studies of performance evolution of offshore teams is still scarce.

Our work contributes to the area of performance measurement of offshore teams, and specifically studies the context of complex large-scale legacy projects, which are particularly challenging in terms of learning and performance. We employ quantitative and qualitative analysis to evaluate the performance evolution of immature remote teams and compare their performance to that of mature teams to better understand the learning targets. We also relate to existing literature on the challenges in learning to explain the changes in performance over time. The context of large-scale software projects involving a large amount of legacy code adds an original perspective of task complexity, which has not been specifically addressed in GSE performance studies before. Furthermore, the performance measurement approach used in our study - which uses task complexity as one of its bases - is more comprehensive than the ones based on traditional metrics (e.g., lines of code or function points).

3 Research Design

To address our research questions, we have conducted an exploratory longitudinal case study [181]. In this section, we describe the case, unit of analysis, and the data collection, preparation and analysis processes.

3.1 The case and unit of analysis

We selected the case studied in this chapter through convenience sampling in consultation with company representatives as a case suitable for studying team learning and performance in a large-scale distributed software development project that involves a large amount of complex legacy code.

The selected case and unit of analysis is a large-scale distributed endeavor associated with the development and maintenance of a large telecommunication software product in Ericsson. Its degree of global distribution increased significantly during the period covered in our investigation. The product originated in Sweden and has evolved for almost 20 years and comprises a considerable amount of complex legacy code. It was subject to many technical and methodological changes over the years, like introducing a new programming language (Java in addition to C++), introducing a new testing technology (TTCN-3) and changing the software development methodology (from plan-driven to agile).

For our investigation, we covered the teams that were still active in October 2016. In total, this involved 188 employees, including 15 software architects and 134 developers working in 19 formal teams. The teams were

5 www.ttcn-3.org
located in Sweden (5 teams), India (10 teams), Italy (1 team), USA (1 team) and Poland (2 teams). Figure 5.2 shows the project evolution in terms of sites. The numbers in the circles represent the number of teams at each site. Offshore locations were added to address the growing demands for resources and to implement market-specific customizations. The teams in India (10 teams) and Poland (1 team) were onboarded relatively recently. The sites in China and Turkey (1 team each) were closed down due to business reasons and are not included in the analysis. The company transferred the main responsibility for the product to India in 2017. Note that 12 developers were onboarded in India during spring 2016 (7 in March 2016 and 5 in May 2016) to replace staff turnover.

The software development teams are cross-functional and use agile practices in their daily work. The teams have four to seven developers and a design lead, who is a senior developer. Due to the scale and level of distribution of the case, the involved project managers use a mix of agile and plan-driven practices to manage and coordinate the work of the software development teams.

The software development teams receive tasks, which can be product customizations, maintenance, product improvements, standardizations of market features and business use cases. Each task resembles an independent project with a specific start and end date and expected results. One or more development teams are assigned to each task. For example, the sched-
ule of a product customization task (PC) varies from one to six months, while business use cases (BUC) may take from three to twelve months.

Scale and distribution of the selected case also influences how the product’s software architecture is managed and how product knowledge is transferred to developers [27]. Architecture management is centralized in Sweden, where a team of software architects ensures the integrity and evolvability of the product architecture. Moreover, the team of architects supports all development teams by responding to architectural questions and by providing feedback on the teams’ work through code reviews. In urgent or particularly complex situations, the software architects may also participate in actual code implementation.

We based our analysis on a subset of the collected data, focusing on PC tasks. We chose PC tasks mainly because it was the only task type for which we were able to track team performance (productivity and autonomy) in a reliable way. PC tasks are driven by external customers and this demands a stricter effort/cost control by Ericsson. Furthermore, PC tasks are more adequate for this study because the immature teams mainly work with this type of task. Finally, PC tasks are challenging (like BUC and maintenance tasks) for immature teams, due to the amount of legacy code and the complexity of the product.

We focused our investigation on the five teams onboarded in India that work with PC tasks to better understand how remote teams perform over time, the challenges they face during their learning process, and how they might be supported to increase their performance. We used data about mature teams located in Sweden, USA and Italy as benchmark. We did not include immature teams from other locations due to two reasons. The teams located in Poland were onboarded too recently to collect enough data points. The teams located in Sweden, Italy and the USA had been onboarded too long ago. This made it difficult to collect reliable data, since part of our measurement approach depended on expert knowledge and people involved with those teams either had left Ericsson or were not able to recall sufficient details regarding the PC tasks.

The PC teams onboarded in India had worked in the case for approximately two years by the time we finalized the data collection process, making it possible to gather data since their inception. Two of the five teams were onboarded in August 2014, one in January 2015 and two more in August 2015. Due to this multi-phased onboarding, the composition of the immature remote teams changed a lot. The onboarding timeline for the 5 teams is summarized in Figure 5.3.

The employed learning process is mainly based on autonomous learning. The new developers received on average three months of formal training about the product (induced learning). Most of the learning is based on doing actual work. Senior developers, mainly software architects located in Sweden, are assigned as mentors [27].
The first two onboarded Indian teams received training and mentoring on-site in Sweden for the initial five months. The other teams received training and mentoring in India for four months, from two visiting senior developers from Sweden.

3.2 Variables

In this chapter, we used the measurement approach detailed in Britto et al. [26], which evolved during the course of this investigation. To address RQ1, we evaluated the performance of the immature remote teams over time, while for RQ2 we measured the performance of mature teams for subsequent comparison. We evaluated performance in terms of team productivity and team autonomy, and used team maturity as dummy variable. The variable are as follows:

- **Task size** $T$ is measured in complexity points and takes into account a task’s extent and complexity to make tasks of different degrees of complexity easier to compare. $T_i$ is the size of task $i$ and $T_{ki}$ is the size of the proportion of task $i$ that was completed by team $k$.
- **Team productivity** $P$ represents the effort that a team spends to complete a complexity point. $P_{ki}$ is the productivity of team $k$ on task $i$:

  $$P_{ki} = \frac{T_{ki}}{E_{ki}} \quad (5.2)$$

  where $E_{ki}$ is the total effort spent by team $k$ on task $i$. 

![Figure 5.3: Onboarding timeline for product customization (PC) teams in India.](image-url)
• **Team autonomy** $A$ represents how independently a development team fulfills a task. $A_{ki}$ is the autonomy with which team $k$ carried out task $i$:

$$A_{ki} = \frac{T_{ki}}{E_{ki}^h + 1} \quad (5.3)$$

where $E_{ki}^h$ is the effort in work hours spent by software architects providing mentoring and help to team $k$ during the fulfillment of task $i$.

• **Team maturity** $M$ represents team maturity. Teams with higher maturity have higher authority and higher independence. We measured team maturity using the authority matrix described by Britto et al. [27], in which teams are classified from A (least authority/maturity) to D (most authority/maturity). Immature teams are classified either as A or B, while mature teams are classified as C or D. To facilitate our analysis, we used only two values: immature (A and B) and mature (C and D). $M_{ki}$ is the maturity of team $k$ before carrying out task $i$.

In addition to task complexity, team productivity and team autonomy, we considered scale in our analysis to address RQ1. The number of developers involved in a task could be used to analyze scale. We identified that there is a strong and statistically significant correlation between the number of developers in a task and the associated task complexity (Spearman’s rank correlation rank = 0.6, p-value = 0.002). Thus, we used in our analysis a variable called task developer ratio, which is defined as follows:

• **Task developer ratio** represents the proportion of developers involved in the fulfillment of a task in relation to task size. $D_i$ of task $i$:

$$D_i = \frac{N_i}{T_i} \quad (5.4)$$

where $N_i$ is the total number of developers involved in task $i$.

### 3.3 Data collection

In total, we collected data from 34 product customization tasks; 24 were carried out by the five immature remote teams (all of them located in India) and 10 were carried by mature teams (two tasks involved a team in Sweden, four a team in USA, and four a team in Italy). The data encompasses tasks carried out between August 2014 (inception of the Indian site)
and October 2016. To collect the data, we conducted semi-structured interviews, archival research and workshops. All interviews and workshops were carried out in person in Sweden, except for one interview conducted with a senior developer located in USA (via Skype).

Five of the 24 tasks carried out by the immature teams involved the participation of mature teams from other locations (tasks D1–D5 in Table 5.2). In the set of tasks carried out by the mature teams, two tasks involved the participation of more than one team (tasks D6 and D7 in Table 5.2). We focused the analysis on the main team in each task. Table 5.2 describes the collaborations involved in the eight distributed PC tasks.

Archival research

A large amount of the data used in our investigation was collected through archival research. The mined data sources, with associated metrics, and variables are described in Tables 5.3 and 5.4. The code review and version control repositories were mined by a representative of Ericsson and the results were handed out as a spreadsheet. The first author of this chapter mined all the files (including the code-related spreadsheets) and aggregated and compiled them in a unique spreadsheet, for posterior data analysis. The aggregated data was verified by means of workshops and interviews, which are described in more detail in the following sub-subsections.

Workshops

We conducted six workshops to assess the complexity of the investigated tasks and verify (sanity check) the data collected through archival research. The workshops took place in 2016, between January and October. They were attended by six to eight software architects. Each workshop took approximately one hour and a half. The participants’ experience in the investigated product ranged from five to 15 years.

In the first workshop with the architects, we defined and piloted the process to measure task complexity. It was consensus among the software architects that existing complexity metrics (e.g., cyclomatic complexity and lines of code) do not account for all the complexity related to product customization tasks in this product. Together with the software architects, we defined the following process to measure task complexity:

1. In the beginning of each workshop, the software architects selected one task and the other tasks were to be measured in relation to it.
2. For each task, the software architects attributed a positive integer number (complexity points) using a planning poker based approach.
3. To check the assigned measures, the software architects consulted solution specification documents, the product architecture specification and the product source code whenever needed.
Table 5.2: Distributed product customization tasks.

<table>
<thead>
<tr>
<th>Distributed Task</th>
<th>Main team Location</th>
<th>Maturity Level</th>
<th>Collaboration Setup</th>
<th>Locations involved</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>India</td>
<td>Immature</td>
<td>Two independent teams</td>
<td>India and Sweden</td>
</tr>
<tr>
<td>D2</td>
<td>India</td>
<td>Immature</td>
<td>Two independent teams</td>
<td>India and Italy</td>
</tr>
<tr>
<td>D3</td>
<td>India</td>
<td>Immature</td>
<td>Virtual team with developers located in India and one software architect located in Sweden</td>
<td>India and Sweden</td>
</tr>
<tr>
<td>D4</td>
<td>India</td>
<td>Immature</td>
<td>Virtual team with developers located in India and one software architect located in Sweden</td>
<td>India and Sweden</td>
</tr>
<tr>
<td>D5</td>
<td>India</td>
<td>Immature</td>
<td>Virtual team with developers located in India and one software architect located in Sweden</td>
<td>India and Sweden</td>
</tr>
<tr>
<td>D6</td>
<td>Sweden</td>
<td>Mature</td>
<td>Temporary team with developers and one software architect, all located in Sweden</td>
<td>Sweden</td>
</tr>
<tr>
<td>D7</td>
<td>Sweden</td>
<td>Mature</td>
<td>Virtual team with developers located in Sweden and one developer located in India</td>
<td>India and Sweden</td>
</tr>
<tr>
<td>Data source</td>
<td>Description</td>
<td>Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task time report spreadsheets</td>
<td>Documents that contain the developers and design leads involved in finished tasks, in addition to the actual effort spent by them to carry out the tasks.</td>
<td>We extracted team setup, number of developers (ND&lt;i&gt;_i&lt;/i&gt;) and task effort (E&lt;i&gt;_i&lt;/i&gt;). We used this data to calculate team productivity (TP&lt;i&gt;_i&lt;/i&gt;), to calculate task developer ratio (TS&lt;i&gt;_i&lt;/i&gt;), and to support the identification of a team’s contribution in distributed tasks.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mentoring time report spreadsheets</td>
<td>Documents that contain the architects and the hours they spent to support teams during the fulfillment of tasks.</td>
<td>From this type of document, we extracted the architects involved with each task and the respective mentoring hours (M&lt;i&gt;_ki&lt;/i&gt;). We used this data to calculate team autonomy (TA&lt;i&gt;_i&lt;/i&gt;).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slot-plan spreadsheets</td>
<td>Documents that contain the planning information related to developers and teams assigned to particular work items. These spreadsheets helped to identify the start and end dates of the tasks.</td>
<td>We used slot-plans to confirm the team setup identified through the time report spreadsheets.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.4: Archival research data sources – part II.

<table>
<thead>
<tr>
<th>Data source</th>
<th>Description</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human resource management repository</td>
<td>Repository that contains information about personnel involved with the selected case.</td>
<td>From this data source, we extracted data regarding the experience of developers and software architects.</td>
</tr>
<tr>
<td>Code repository</td>
<td>Configuration management is carried out using GIT in the selected case.</td>
<td>We extracted lines of code associated with each task, which were used to support the sanity check carried out regarding task complexity ($C_i$).</td>
</tr>
<tr>
<td>Product architecture specification</td>
<td>Document that contains the description of the product architecture and its main components.</td>
<td>It was used to support the measurement of task complexity ($C_i$).</td>
</tr>
<tr>
<td>Solution specifications</td>
<td>Documents that contain details about solution design.</td>
<td>It was used to support the measurement of complexity ($C_i$).</td>
</tr>
</tbody>
</table>
The task complexity measurement took place from the second to the sixth workshop with the software architects. The results were recorded in a spreadsheet.

In the sixth workshop with the architects, we also conducted a data sanity check to validate both the assessed task complexity and the plots about performance evolution. It was based on summaries and plots of the collected data, including lines of code and task complexities. We asked the architects to provide explanations for the outliers. They realized they had not accounted for non-functional testing in two product customization tasks. As a consequence, they increased the amount of task complexity points associated with these two tasks. We kept notes about the developers’ thoughts regarding the performance evolution results, which were mainly used to address RQ1.

An additional workshop took place with the participation of five developers from the immature remote teams (located in India). The workshop took one hour and 20 minutes and was held in November 2016. We conducted this workshop with developers who spent three months in Sweden as part of the fulfillment of one product customization task. The experience of the participants in the investigated product ranged from 1.5 to 2.5 years.

We asked the participants to provide information about the challenges they faced during the process of acquiring the knowledge required to work in the project under investigation. The developers were asked to write down, independently of each other, the challenges that, in their opinion, impacted their learning processes. After 10 minutes, individual opinions were discussed within the group. The results were used to find explanations for performance evolution and were aggregated in a report that was verified by the participants. Quotes from this report are used in the discussion of the results.

Semi-structured interviews

We conducted individual semi-structured interviews with three software architects in 2016 between April and October. Each interview took approximately one hour. The interviewees had from 5 to 15 years of experience in the investigated product.

The main objectives of the interviews were:

- To identify the teams’ contribution to the distributed product customization task’s, and how much mentoring and help these teams received by software architects. We used this information to adjust $T_i$ for tasks that involved several teams ($T_{ki}$).
- To measure team maturity ($M_{ki}$) using the authority matrix presented by Britto et al. [27]. We asked each architect to classify the teams supported by him or her in the tasks under investigation.
• To collect additional information about the challenges the immature remote teams faced to acquire the knowledge required to work in the tasks under investigation.

We also conducted one semi-structured interview via Skype with a design lead located in the USA. He had more than 10 years of experience in the investigated product by the time of our investigation. The interview took 30 minutes and was conducted in September 2016. The main goal of this interview was to get the view of an experienced developer about the challenges immature remote teams face when acquiring the knowledge required to work in the product under investigation.

In all interviews, we took notes to facilitate posterior analysis. The notes included key points related to the questions that were posed during the interviews. The notes were discussed with the respective interviewees, to ensure that they reflected what was discussed during the interviews. The team maturity for each team, as assessed by the architects, was recorded in a spreadsheet.

3.4 Data preparation

Data preparation represents the process of logging data regarding the time when it was collected; checking the accuracy and reliability of the collected data; transforming the collected data; and consolidating it in a centralized structure [198].

In this case study, we used a significant amount of data, which was extracted using different methods, had different formats and came from different sources. Furthermore, some of the used variables were not directly measurable. To mine, log, transform, sanitize and aggregate the data, we developed various R7 scripts, which were used to create a centralized spreadsheet.

3.5 Data analysis

The data analysis was conducted in three steps. First, we aimed at evaluating performance of remote teams onboarded in the studied product over time. To do so, we first plotted LOWESS plots (Locally Weighted Scatterplot Smoothing) [34] to identify how the performance of the immature remote teams evolved in their two first years (RQ1). This approach is more adequate to identify trends in a dataset with a small number of observations than traditional linear regression analysis, which demand a considerable amount of observations [81, 143]. Second, we employed a Wilcoxon-Mann-Whitney test to identify whether there is a significant dif-
ference between the performance of immature teams working in tasks with low and high task developer ratio, respectively. Third, we used the Vargha-Delaney A measure \[161\] to calculate the effect size of task developer ratio regarding the performance of the immature teams. Fourth, we calculated the difference ratio between the performance of immature teams working in tasks with different levels of scaling based on descriptive statistics. Finally, we plotted boxplots to enrich the data analysis.

In the second step of the data analysis, we focused on identifying the magnitude of differences in performance of the immature remote teams and the mature teams through analyzing the collected quantitative data (RQ2). To do so, we first applied a Wilcoxon-Mann-Whitney test \[180\] to identify whether there is a significant difference between the immature and mature teams. After that, we used the Vargha-Delaney A measure \[161\] to calculate the effect size of maturity regarding the performance of teams. Third, we calculated the ratio between the performance of mature and immature teams based on descriptive statistics. Finally, we plotted boxplots to enrich the data analysis.

In the third step of the data analysis, we analyzed the qualitative data collected through interviews and workshops to identify the explanations of the performance evolution and understand the challenges faced by the immature teams during their learning processes. We also looked at practices already employed by Ericsson to support the learning processes of immature remote teams. To do so, we employed a light-weight qualitative analysis approach \[167\] and aggregated the results of the interviews and workshops to identify challenges and practices.

4 Results

In the following subsections, we present the results of our investigation, grouped by research questions.

4.1 RQ1 – Performance evolution of immature remote teams

To analyze how the performance of the newly onboarded remote teams varied during the first two years of their engagement, we created LOWESS plots (Figures 5.4 and 5.8) to visualize productivity and autonomy changes on a timeline based on task completion date.

Team productivity

Figure 5.4 shows team productivity over time. The x-axis shows the completion date for each investigated product customization task, while the y-
Figure 5.4: Team productivity evolution for immature teams based on task completion date (24 tasks).
axis shows the productivity for the respective tasks\textsuperscript{8}. The trend line shows that the productivity of the immature remote teams increased initially and peaked after about 6 months (August 2015 – 1508) after completion of the first task (Task-1). Productivity then decreased gradually during about 8 months to finally return to the top-level of the first year at the end of the second year. To seek the explanation for the changes in productivity, we organized a workshop with software architects. The participants consented that the productivity drop is likely to be related to four factors: the frequent onboarding of new developers that aimed at increasing the manpower and replacing staff attrition, remote mentoring, distribution and high task developer ratio (the proportion of developers per task complexity).

As shown in Figure 5.3, the Indian site started with two product customization teams (five and six developers, respectively) in August 2014. In January 2015, one more product customization team was added (a total of 18 developers, six per team), followed by another two teams in August 2015 (a total of 28 developers, three teams with five developers each, one with seven developers and one with six developers). Furthermore, 10 developers were added to replace developers that left the project (three left in the first half of 2015 and seven in the second half of 2015). The onboarding of new developers to existing teams meant that there were a large number of new developers who needed training and mentoring, which may explain why the productivity of the investigated teams did not improve as expected.

A factor that seems to have amplified the challenges imposed by staff attrition is the employed mentoring approach. The largest productivity improvement occurred when the two initially onboarded immature remote teams were trained and mentored on-site in Sweden (between August 2014 and January 2015) and when two senior developers from Sweden traveled to India to provide on site mentoring in India (between February and May 2015). From June 2015 onwards, the immature teams received only remote mentoring by software architects located in Sweden. This matches the period wherein their productivity goes down in Figure 5.4.

Regarding distribution, it is possible to see in Figure 5.4 that the immature teams had their best performance when they were not required to work on a task in collaboration with teams from other locations (tasks 1, 3, 9, 17 and 18). To identify whether there was a significant statistical difference between the productivity of the immature teams in collocated and distributed tasks, we defined the following null hypothesis:

\begin{itemize}
  \item H\textsubscript{1} – The productivity of immature remote teams does not differ significantly between collocated and distributed.
\end{itemize}

\textsuperscript{8} In Figures 5.4, 5.6, 5.8 and 5.10, the tasks with blue label (1, 3, 9, 17 and 18) were conducted in a distributed fashion. They are described in more details in Table 5.2 (D1-D5 respectively).
Table 5.5 presents descriptive statistics regarding team productivity in relation to distribution, while Figure 5.5 shows the associated boxplot.

Table 5.5: Descriptive statistics of team productivity in relation to distribution.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Col-Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collocated</td>
<td>19</td>
<td>29.12</td>
<td>31.60</td>
<td>4.86</td>
<td>8.98</td>
<td>46.44</td>
</tr>
</tbody>
</table>
| Distributed | 5       | 9.35   | 8.96   | 5.37      | 3.44  | 16.81     | Mean: 3.11  
|             |         |        |        |           |       | Median: 3.52 |

The results show that the average productivity of the immature remote teams for collocated tasks is 3.11 (median 3.52) times higher than for distributed tasks.

We applied a Wilcoxon-Mann-Whitney test to evaluate H1 resulting in a p-value of $p = 1.37 \times 10^{-3}$, which means that $H_1$ is rejected with high statistical power, i.e. the difference is highly significant.

To further identify the impact of distribution on team productivity, we used the Vargha-Delaney A measure to calculate the effect size of task developer ratio on team productivity. The resulting A measure is equal to 0.94, which indicates that the immature teams had a better productivity for collocated tasks 94% of the times.

To investigate the relationship between productivity and task developer ratio, we plotted Figure 5.6, wherein the x-axis shows the task’s scaling scores and the y-axis the corresponding productivity. The plot shows that the performance drop is highly related to task developer ratio. Figure 5.6 also shows that the immature teams had their best performance when the proportion between the number of developers and task complexity (task developer ratio) is $\leq 0.2$ (two developers per 10 complexity points).

Given the difference in productivity due to the proportion of developers per complexity points, we further investigated the significance of this finding. To do so, we grouped the tasks into two groups: low scaling group, including tasks with task developer ratio $\leq 0.2$; and high scaling group, with tasks with task developer ratio $> 0.2$. For our statistical analysis, we defined the following null hypothesis:

- $H_2$ – The productivity of immature remote teams does not differ significantly between tasks with high and low task developer ratio.

Table 5.6 presents descriptive statistics regarding team productivity in relation to task developer ratio, while Figure 5.7 shows the associated boxplot.
Figure 5.5: Boxplot of team productivity and distribution.

Table 5.6: Descriptive statistics of team productivity in relation to task developer ratio.

<table>
<thead>
<tr>
<th>Scaling</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>15</td>
<td>31.10</td>
<td>33.73</td>
<td>13.30</td>
<td>3.44</td>
<td>46.44</td>
<td>Mean: 2.10</td>
<td>Median: 2.06</td>
</tr>
<tr>
<td>High</td>
<td>9</td>
<td>14.84</td>
<td>16.41</td>
<td>5.37</td>
<td>8.96</td>
<td>21.87</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.6: Team productivity versus task developer ratio for immature teams (24 tasks).
The results show that the average productivity of the immature teams for tasks with low task developer ratio is 2.10 (median 2.06) times higher than in tasks with high task developer ratio (> 0.2).

![Boxplot of team productivity and task developer ratio.](image)

We applied a Wilcoxon-Mann-Whitney test to evaluate H2 resulting in a p-value of $p = 6.71 \times 10^{-3}$, which means that H2 is rejected with high statistical power, i.e. the differences in productivity for the two task developer ratio groups is highly significant.

To further identify the impact of task developer ratio on team productivity, we used the Vargha-Delaney A measure to calculate the effect size of task developer ratio on team productivity. The resulting A measure is equal to 0.83, which indicates that the immature teams had a better productivity when the task developer ratio was low ($\leq 0.2$ developers per 10 task complexity points) in 83% of the times.
Note that the main outliers in Figure 5.6 (Task-3 and Task-9) are distributed product customization tasks, wherein the immature remote teams worked with one team in Italy and one team in Sweden, respectively.

**Team autonomy**

![Team autonomy evolution for immature teams based on task completion date (24 tasks).](image)

Figure 5.8 shows team autonomy over time. The x-axis shows the completion date for each investigated product customization task, while the y-axis shows team autonomy for the corresponding tasks. Compared to Figure 5.4, we can see that the variation of team autonomy over time is smaller than the variation of team productivity. As for productivity, we can observe an initial increase in autonomy. After that, we have a year-long
period of stagnation. In the last five months (from June 2016 onwards), we can finally see a further increase in autonomy.

To explain these results, we held a discussion with the software architects. Similarly to productivity, the initial increase in team autonomy was associated with on-site mentoring, while the lack of improvement or stagnation in the middle was said to be likely caused by remote mentoring, the frequent onboarding of new developers (increase of manpower and replacement of staff attrition), distribution and task developer ratio. However, the effect of these factors is smaller for autonomy than for productivity.

At first glance, the stagnation period seems counter-intuitive. It is fair to assume that the addition of new developers together with remote mentoring should have called for more mentoring hours because of the inefficiency of remote communication. In contrast, we found that the remote teams perceive the mentoring functions less accessible and often do not reach out for help that often. This might also explain why productivity went down, since the knowledge gap that was compensated by the short-term involvement of software architects later consumed more effort from the immature teams.

Regarding distribution, there is no clear difference regarding team autonomy in Figure 5.8. To confirm whether there was not a significant statistical difference between the autonomy of the immature teams in collocated and distributed tasks, we defined the following null hypothesis:

- \( H_3 \) – The autonomy of immature remote teams does not differ significantly between collocated and distributed.

Table 5.7 presents descriptive statistics regarding team autonomy in relation to distribution, while Figure 5.9 shows the associated boxplot.

Table 5.7: Descriptive statistics of team autonomy in relation to distribution.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Col</th>
<th>Dist.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collocated</td>
<td>19</td>
<td>0.53</td>
<td>0.42</td>
<td>0.30</td>
<td>0.22</td>
<td>1.32</td>
<td></td>
</tr>
<tr>
<td>Distributed</td>
<td>5</td>
<td>0.31</td>
<td>0.22</td>
<td>0.15</td>
<td>0.19</td>
<td>0.56</td>
<td>1.69</td>
</tr>
</tbody>
</table>

The results show that the average autonomy of the immature remote teams in collocated tasks is 1.69 (median 1.90) times higher than in distributed tasks. However, after applying Wilcoxon-Mann-Whitney test to evaluate \( H_3 \), we identified that this difference is not statistically significant (\( p\) – value = 0.1).
Figure 5.9: Boxplot of team autonomy and distribution.
To investigate the relation of task developer ratio and team autonomy, we plotted task developer ratio on the x-axis and team autonomy data for the corresponding tasks on the y-axis (see Figure 5.10). Similarly to the productivity analysis, we carried out a statistical analysis for the relationship between team autonomy and task developer ratio with the following null hypothesis:

- **H4** – The autonomy of immature remote teams does not differ significantly between tasks with high and low task developer ratio.

![Figure 5.10: Team autonomy versus task developer ratio for immature teams (24 tasks).](image)

Table 5.8 presents descriptive statistics regarding team autonomy in relation to task developer ratio, while Figure 5.11 shows the associated boxplot.
Table 5.8: Descriptive statistics of team autonomy regarding task developer ratio.

<table>
<thead>
<tr>
<th>Scaling</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>15</td>
<td>0.48</td>
<td>0.42</td>
<td>0.28</td>
<td>0.22</td>
<td>1.32</td>
<td>Mean: 0.98</td>
<td>Median: 1.17</td>
</tr>
<tr>
<td>High</td>
<td>9</td>
<td>0.49</td>
<td>0.36</td>
<td>0.31</td>
<td>0.20</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results show that there is no clear difference between the autonomy of the immature remote teams in tasks with low task developer ratio and tasks with high task developer ratio (mean ratio 0.98; median ratio 1.17). However, after applying Wilcoxon-Mann-Whitney test to evaluate H4, we identified that this difference is not statistically significant (p-value = 0.79).

4.2 RQ2 – Performance gap between immature remote teams and mature teams

With RQ2, we aimed at identifying the magnitude of the performance gap that an immature remote team need to overcome mainly through learning. Although the research on performance gaps between experts and novices is well established, this was of particular importance to us to better understand the learning targets, i.e. how large is the gap that the new teams shall overcome. For our statistical analysis, we formulated the following null hypotheses:

- **H5** – The productivity of immature remote teams and mature teams do not differ significantly.
- **H6** – The autonomy of immature remote teams and mature teams do not differ significantly.

**Team productivity**

Table 5.9 presents descriptive statistics regarding team productivity, while Figure 5.12 shows a boxplot of team productivity in relation to maturity.

The results show that the average productivity of mature teams is 3.67 (median 3.73) times higher than the productivity of immature remote teams. Furthermore, even considering that the productivity of both immature and mature teams varied over time, the results show that the lowest productivity of mature teams (58.41) is still higher than the highest productivity of immature remote teams (46.44).
Figure 5.11: Boxplot of team autonomy and task developer ratio for immature teams (24 tasks).

Table 5.9: Descriptive statistics of team productivity in relation to maturity.

<table>
<thead>
<tr>
<th>Maturity</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
<th>Mature</th>
<th>Immature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immature</td>
<td>24</td>
<td>25.07</td>
<td>26.38</td>
<td>13.56</td>
<td>3.44</td>
<td>46.44</td>
<td>Mean: 3.67</td>
<td></td>
</tr>
<tr>
<td>Mature</td>
<td>10</td>
<td>92.12</td>
<td>98.42</td>
<td>23.84</td>
<td>58.41</td>
<td>120.07</td>
<td>Median: 3.73</td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.12: Boxplot of team productivity and maturity.
To evaluate $H_5$, we applied a Wilcoxon-Mann-Whitney test resulting in a p-value of $p = 1.53E-08$, which means that $H_5$ can be rejected with high statistical power, i.e. the difference in productivity between immature and mature teams is highly significant.

To further identify the impact of maturity on team productivity, we used the Vargha-Delaney A measure to calculate the effect size that maturity has on team productivity. When comparing two populations (immature and mature teams in our case), the A measure (a number between 0 and 1) expresses how often one population outperforms the other. In our case, the resulting A measure is equal to 1, which indicates that the mature team always had higher productivity than the immature teams (in the context of the analyzed dataset).

**Team autonomy**

Table 5.10 presents descriptive statistics regarding team autonomy, while Figure 5.13 shows a boxplot of team autonomy in relation to maturity.

Table 5.10: Descriptive statistics of team autonomy and maturity.

<table>
<thead>
<tr>
<th>Maturity</th>
<th>Observ.</th>
<th>Mean</th>
<th>Median</th>
<th>Std. Dev.</th>
<th>Min</th>
<th>Max</th>
<th>Mature Immature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immature</td>
<td>24</td>
<td>0.48</td>
<td>0.37</td>
<td>0.28</td>
<td>0.19</td>
<td>1.32</td>
<td>Mean: 2.27</td>
</tr>
<tr>
<td>Mature</td>
<td>10</td>
<td>1.09</td>
<td>0.93</td>
<td>0.73</td>
<td>0.35</td>
<td>2.83</td>
<td>Median: 2.54</td>
</tr>
</tbody>
</table>

The results for team autonomy show that the mature teams are on average 2.27 (median 2.54) times more autonomous than the immature remote teams.

Similarly to the productivity analysis, we also applied a Wilcoxon-Mann-Whitney test to evaluate $H_6$ resulting in a p-value of $p = 0.003$, which means that $H_6$ can be rejected with high statistical power, i.e. the difference in autonomy between immature and mature teams is highly significant.

We used the Vargha-Delaney A measure to calculate the effect size of maturity on team autonomy. The resulting A measure is equal to 0.83, which indicates that the mature teams had higher autonomy than the immature teams in 83% of the cases (in the context of the analyzed dataset).

5 Discussion

The performance of the immature remote teams in our study was considerably lower than the performance of the mature teams. Furthermore, after
Figure 5.13: Boxplot of team autonomy regarding maturity.
a short period of increased performance, the teams struggled to evolve further. To identify explanations for such results, we conducted semi-structured interviews and workshops with software architects and remote developers, to complement our archival research. We identified four main challenges faced by the immature remote teams during their learning processes, which may have impacted their learning process and performance evolution: complexity of the product and the technology stack\(^9\), distance to the main source of product knowledge, lack of team stability and training expectation misalignment.

The challenges are described in more detail in the following paragraphs and supported by quotes from the semi-structured interviews and the individual parts from the workshop with developers (see Section 3.3). For each quote, the acronym in parentheses indicates the source of the quote: SA indicates quotes from software architects, MD indicates quotes from developers from mature remote teams and ID indicates quotes from developers from immature remote teams.

**Complexity of the product and technology stack**

In the investigated case, new teams were onboarded when the product had evolved for 17 years. This means that a considerable amount of complex legacy was developed during this period, which made the learning process harder, especially regarding the acquisition of product knowledge.

The product investigated in this case study is very complex, includes several subsystems and comprises millions of lines of code, including a significant amount of legacy code. Although not all parts are equally complex, there are many subsystems and components that are too complex to be handled by immature teams. In addition, the product also involves many different technologies, which in some cases are company specific in-house developments. This means that, in addition to a large amount of legacy code, the immature remote teams also need to deal with a complex set of technologies, making the process of acquiring the required knowledge particularly difficult.

All participants stated explicitly that the product size and complexity makes it challenging to learn and to work with:

---

\(^9\) By technology stack, we mean the set of technologies used in the project, such as programming languages and frameworks, database, testing frameworks and code review tool.
Other immature developer highlighted that they mainly worked with the product testing framework and do little work inside the application code. Due to the product’s complexity, it is not uncommon that immature developers are more involved in developing the testing framework than evolving the application code. Immature developers are more likely to start with non-core areas, to avoid schedule overruns. Moreover, for each new feature that is added to the product, there are hundreds of legacy features that need to be verified through regression testing. This means that the majority of the development effort is related to updating test cases in the used testing framework. However, this emphasis on testing seems to impact the motivation of the immature developers:

**Quote 5 (ID)** – “We spent 70% of our time working with the testing framework. I believe that we should do more about application code to learn more about the product.”

One immature developer pointed out the complexity of the technology stack. Some of these technologies are Ericsson-specific, which reduces the sources from where the immature teams can acquire the required knowledge (e.g., it is not possible to use commonly used sites like Stack Overflow\(^\text{10}\) to learn about these technologies). However, it is worth noting that this type of competence is available locally at the remote site; some of the onboarded developers already had worked with the same technologies in other products at Ericsson before moving to the product investigated in this case study.

\(^{10}\) www.stackoverflow.com
Distance to the main source of knowledge

Existing literature shows that it is more difficult to communicate and coordinate in distributed projects [68, 89, 90]. While this is true for experienced developers, it is fair to assume that remote knowledge transfer and learning is even more challenging. In the investigated case, there was always some level of distribution (e.g. remote mentoring), even in tasks that were carried out by just one team.

In the case, the main responsible for ensuring code quality and architectural evolvability is the team of software architects located in Sweden. Furthermore, the software architects, together with some senior developers located in Sweden, are also the main sources of product knowledge and work as mentors for the immature remote teams. Considering the geographical, temporal and socio-cultural distances between the immature teams and the software architects, it is particularly challenging for the immature teams to acquire the knowledge required. These issues are recognized by software architects and developers as well:

Quote 7 (SA) – “I believe that the reason the immature teams in India are struggling is because the product is complex to learn and it is very hard for us to help them just using code reviews, emails and Skype.”

Quote 8 (ID) – “Since we didn’t get that much training about the product before starting working, we have to keep learning stuff. The two main sources we have are the software architects and documentation. However, the architects are in Sweden, which complicates the communication with them, and the documentation sometimes does not have everything and it is hard to find the required knowledge.”

One immature developer commented about the code review process, which is not only used to ensure code quality, but also for mentoring. Although the immature developer acknowledges that code reviews is a good approach to learn about the product, geographical distance and especially temporal distances affect how the process is organized in practice:
The immature remote teams and the software architects communicate face-to-face via videoconferencing only for critical tasks and during critical phases; status meetings are held twice per week for high priority tasks and to deal with task software design matters. The immature developers highlighted that they would like to have more videoconference meetings to support their learning process:

**Quote 9 (ID)** – “We learn a lot from the feedback we get from code reviews. However, sometimes it takes quite a while to get the feedback, because the experienced reviewers are located either in Sweden or the USA.”

**Quote 10 (ID)** – “In general, we mainly communicate via Skype, Gerrit and email. Videoconferences just take place more regularly for high-priority PCs and during the design phase. This makes things harder for us.”

**Quote 11 (ID)** – “I believe that we should have more videoconferences. For example, during the time we spent in Sweden, things worked much better because we could talk to the architects any time to clarify any doubt about the product. More frequent videoconferences would allow us to have more frequent feedback.”

Our finding is in line with the previous research that suggests that remote and onshore developers go through a different learning curve due to the distance to the main source of knowledge [211].

**Training expectation misalignment**

The studied Ericsson business unit employs a learning process mainly based on autonomous learning, i.e. new developers will not have the entire required knowledge before starting real work. While this approach worked well when onboarding new teams in Sweden, USA, Italy and Poland, it seems to be a challenge for new immature teams in India. Although the teams located in India received more formal training than the teams located in other locations (e.g., USA, Sweden, Italy and Poland), their general perception is that it was still insufficient.

The data we collected through the interviews shows that the training expectations of the Indian developers and the type of training/mentoring offered by Ericsson were not aligned. One possible explanation for this could be rooted in the cultural differences. According to Nicholson and Sahay [163], who conducted a longitudinal study to identify political and
cultural issues in globally distributed projects, the education system in India is more traditional and emphasizes more discipline than the ones in the European countries. Indian developers are more used to traditional rote learning and “receiving” information. The learning process employed in the project is mainly based on autonomous learning and expects proactive learners. This might have clashed with the expectations of the Indian developers. This misalignment or perceived “lack of training” seems to have impacted their learning process negatively:

**Quote 12 (ID)** – “We just get a very short training about the product before starting doing the work. It is more like a helicopter view and we just learn something when we start working. In addition, we don’t get enough training about the used technologies. This for sure impacts how we learn and how we work.”

**Quote 13 (ID)** – “Once we were asked to code the test cases of a PC using a particular technology that we have never gotten trained or worked with before. Thus, we had to coordinate with people from other Indian site, which made it harder for us to acquire the required knowledge and to perform our work in a better way.”

**Lack of team stability**

Although not highlighted explicitly by any of the interviewees, during the archival research we learned that the remote teams were very unstable in the first two years of their engagement. The reason for that was the onboarding of several new developers during this period, to increase the available manpower and to replace staff attrition. In general, the new developers were integrated in existing teams to facilitate the onboarding and learning. In some cases, teams were merged to achieve the needed capacity for the development of larger and more complex product customization tasks. Team instability is a known hindering factor for group learning and affects the performance of development teams negatively [160].

When we investigated the immature teams, we also found a few cases wherein the management created temporary virtual teams by combining developers from the Indian site and software architects located in Sweden, to ensure the timely delivery of high priority product customizations. In such situations, software architects were mainly responsible for developing the application code, while the Indian developers took care of the testing framework.
The default strategy was to let the immature remote teams do the application code. However, when they failed to get it right, the software architects took it over to avoid schedule overrun. In such situations, it would take longer time to explain to the immature remote teams what they would need to do. This made the learning process of immature remote teams more difficult, because they could not exercise group learning and they were not able to deal with application code very often.

5.1 Implications for research

In this chapter, we have employed a novel approach to investigate learning and performance of remote immature teams in large-scale distributed projects. However, we have investigated just one case. Further research is necessary to strengthen the empirical evidence.

By the time we conducted this study, the immature remote teams had been onboarded slightly more than two years ago, which limited the number of tasks (observations) we could analyze. We believe that investigating a longer period of time might enrich our understanding about the performance evolution of remote immature teams. For example, it might allow us to study when immature teams will reach and sustain the same level of performance of mature teams. Due to distance, socio-cultural factors or attrition, remote teams might take a long time or never reach the performance level of (local) mature teams [50].

Finally, learning is just one component of onboarding [136], which represents the strategies and practices organizations use to support employees in their acquisition of the knowledge, skills, and attitudes required to successfully perform their work. The learning and performance evolution of teams is affected by all components of onboarding, i.e. more research on this topic may benefit research and practice.

5.2 Implications for practice

Based on the results and our observations regarding this case study, we have five recommendations for facilitating the learning process of immature teams and tracking their learning and performance progression.

Recommendation 1 – Take into account cultural differences when designing the learning process

Our results as well as related literature shows [213] that the lack of cultural awareness seems to impact the efficiency of the learning process of immature remote teams. On one hand, the learning process that they are still going through was mainly designed in Sweden, assuming a local learn-
ing background based on autonomous learning. On the other hand, the immature developers in India expected more training and frequent feedback from the mentors (software architects), based on their learning background.

This can be addressed by tailoring the learning process to accommodate cultural differences and by aligning expectations from both sides involved in the learning process. Ericsson has improved their learning process by increasing the amount of more traditional training hours for the teams in India, as compared to the teams onboarded in other locations. More frequent meetings via videoconferences between the teams in India and the mentors in Sweden have helped to improve the alignment of expectations.

**Recommendation 2 – Ensure collocated mentoring to support the learning process**

Existing literature shows [213] that awareness across sites is difficult to achieve. Thus, collocation helps immature remote teams to learn who knows what and who the experts are. Collocation also facilitates the collaboration between teams and mentors, which shortens feedback cycles.

In the investigated case, an initial group of newly onboarded developers spent five months in Sweden. Afterwards, two senior software architects spent four additional months providing on-site mentoring in India. In this case, it was impossible to prolong the architects’ stay in India and also to send the developers from India to Sweden more often or for longer periods of time.

To address these limitations, Ericsson started to train core developers from India to become software architects. This was rather challenging, since it takes a long time to acquire the knowledge required to work as a mentor. In our case, two developers were selected as candidates after two years of working in the product. During the time of this investigation, these candidates underwent on-site training in Sweden. While this cannot replace collocation with senior developers and software architects, it can help to disseminate the knowledge locally and facilitate the learning process of newly onboarded developers, in particular in situations of high staff attrition.

**Recommendation 3 – Facilitate group learning**

Group learning is associated with better team performance [64]. Thus, it is very important to facilitate group learning by keeping teams as stable as possible. While stability of the studied teams in India was difficult to ensure because of continuous expansion and addition of new developers and teams, management decided to address team instability after becoming aware of the results presented herein.
Recommendation 4 – Use code reviews to support the learning process

One of the key assets of the learning process in the studied case were code reviews, which are maintained in Gerrit, a special code review software tool. The usage of this tool, empowered by the prevalent open source mindset, encourages knowledge sharing, facilitating the collaboration between immature developers and software architects acting as mentors.

Recommendation 5 – Track the progress of immature teams in a more systematic way

Besides facilitating the learning process of immature remote teams, it is also fundamental to track their progress and identify whether there are deviations that need to be addressed. To do so, it is mandatory to define performance goals and instruments to assess teams performance against defined goals. By doing so, organizations can identify whether teams are on track or need more or less formal training or mentoring hours. To track the progress of immature teams, organizations could use the measurement approach for productivity and autonomy employed in this chapter.

To track productivity, organizations could simply compare the productivity of the teams, measured using our approach, with the defined goals. To track autonomy, organizations could use the authority matrix we developed in a previous work [27]. The authority matrix is presented in Figure 5.14. It has a learning and mentoring curve that shows how the responsibilities of software architects and development teams change as teams get more mature. In its y-axis, there are the four main activities that software development teams and architects are involved in the investigated case, while the x-axis has the four defined maturity levels.

6 Threats to validity

The validity threats associated with our investigation are discussed using the categories reliability, internal, construct and external validity described by Runeson and Höst [181]. Furthermore, we also discuss conclusion validity [198], since we applied statistical inferences in this investigation.

Reliability is related to the repeatability of a study, i.e. how dependent are the data and their analysis on the researchers who collected and analyzed it [181]. We minimized this threat by involving several researchers in the design and execution of our investigation. Furthermore, our observations and findings were verified with the company representatives to avoid false interpretations. We also designed and followed an explicit case study protocol, following the guidelines by Runeson and Höst [181].

However, the approach we used to assess task complexity was highly dependent on the involved software architects. It might therefore be very
Figure 5.14: The authority matrix [27].
hard to obtain the same values with other informants. Since we used an iterative process for task complexity assessment (see Section 3.3), we would expect similar results, though. The same applies to the approach used to adjust team productivity and autonomy of distributed product customization tasks.

**Internal validity** is related to factors that researchers are unaware of or cannot control regarding their effect on the variables under investigation [181]. The facets of performance we investigated are influenced by several confounding factors. We made an attempt to mitigate this threat by interviewing people with different roles and by using existing literature (data triangulation). Regarding the qualitative part of our research, the main internal validity threats are investigator bias and interviewee bias. To mitigate these threats, three researchers were involved with the design of the interview and workshop guides (investigator triangulation). We mitigated the second threat by interviewing people with different roles (data triangulation).

**Construct validity** reflects how well the measures used actually represent the constructs the study intends to measure [181]. The main threat to construct validity is that we used only one method to measure a construct. To mitigate this threat, we collected data from different sources (data triangulation). Furthermore, we conducted a sanity check together with the software architects to validate our measures (task complexity, team autonomy and task developer ratio). To support these sanity checks, we used plots and descriptive statistics to identify inconsistencies.

Since task complexity was obtained using expert judgment, this is a potential threat to construct validity (expert bias). This threat was mitigated by involving several experts with many years of experience in the product under investigation and employing an iterative assessment approach (based on planning poker). Existing literature shows that group discussions lead to better effort estimates than expert judgment [154].

**External validity** is concerned with the generalization of the findings [181]. Since we employed the case study method, our findings are strongly bound by the context of our study. In addition, the investigated case involved only one product in one company. To mitigate this threat, we made an attempt to describe the context of our study in as much detail as possible. This makes it easier to relate the present case to similar cases.

**Conclusion validity** is concerned with the correctness of conclusions regarding relationships in the analyzed data [198]. The main threats to conclusion validity are the low reliability of measures (the amount of noise related to a measure) and low statistical power. To mitigate the first threat, we conducted a sanity check of the collected data, as mentioned above. Regarding statistical power, we were limited to data covering product customization tasks carried out by the immature remote teams during two years. It was not possible to collect a similar amount of data for the mature
teams due to the fact the data regarding the mature teams was not fully available; most product customization tasks carried out by mature teams were too old.

7 CONCLUSIONS AND FUTURE WORK

This chapter presents the results of a case study conducted in Ericsson on the learning and performance evolution of immature remote teams in a large-scale project involving significant amounts of legacy code. The overall conclusion is that scale, global distribution and complex legacy code affect the learning process and performance evolution of the investigated remote teams.

Regarding RQ1, our results suggest that the newly onboarded remote teams did not show the expected performance improvements over time. The frequent onboarding of new developers to either increase manpower or to replace staff attrition, imposed high learning demands. However, the learning process of remote teams was challenged by the following: the complexity of the product and technology stack, the distance to the main source of knowledge, lack of team stability and training expectation misalignment. The complexity of the product and associated technologies make the learning process hard, which was amplified by the fact that only remote mentoring was available after nine months (geographical and temporal distance challenges). A lack of cultural awareness appears to have impacted the learning as well, since training designed by Swedes did not fulfill the expectations of the teams based in India. Finally, frequent onboarding led to high team instability.

In relation to RQ2, we identified that the productivity of the investigated immature remote teams was on average 3.67 times lower than the productivity of mature teams. Although this the difference between novices and experts is not necessarily surprising, the gap for the newly onboarded teams to overcome is considerable and given the lack of performance improvement in the first two years, unlikely to be reached in the nearest future. We also identified that the immature remote teams were on average 2.22 times less autonomous than the mature teams. The differences are statistically highly significant and the effect sizes are high.

The results of this chapter indicate that onboarding in globally distributed projects that involve a considerable amount of complex legacy code must be planned well ahead. It might take a long time until immature teams acquire all knowledge required to perform as well as mature teams. Organizations must be prepared to provide extended periods of mentoring by expensive and potentially scarce resources (e.g., software architects).
In this chapter, we covered approximately two years of data. Covering a longer period of time might provide further insights into the learning processes and performance evolution of remote teams. Therefore, we plan to continue collecting data about this case. Since learning is just one of the components of onboarding, we are also investigating how to study further components of onboarding, such as recruitment and mentoring.
ONBOARDING SOFTWARE DEVELOPERS AND TEAMS IN THREE GLOBALLY DISTRIBUTED LEGACY PROJECTS: A MULTI-CASE STUDY

1 INTRODUCTION

Day after day, companies must deal with different challenges to survive in an increasingly competitive market. For software companies, the source of competitive advantage has always been associated with the competent resources due to the knowledge-intensive nature of the work. Therefore, recruiting and onboarding new employees is one of the key areas of success. Onboarding (also known as organizational socialization) is the process of supporting new employees regarding their social and performance adjustment to their new job [9]. In the context of software development, many reasons may lead to onboarding of new developers such as: i) to replace retired developers; ii) to replace developers that left or will leave the company; iii) to scale up the number of developers in response to the growth of the number of customers; iv) to incorporate new people that may bring new ideas and thus help a company to innovate; or v) to take over the work from original developers (also a type of replacement) in order to free up the experienced developers for starting something new. In response, companies must recruit new people, or, in other words, start the “onboarding” process, by which newcomers make the transition from being organizational outsiders to being insiders [203]. Effective onboarding helps new employees to learn attitudes, knowledge, skills and behaviors required to work effectively [9, 11, 37, 120].

The onboarding process can be either formal or informal [133, 134, 225]. In an informal onboarding, new employees learn about their new job without an explicit organizational plan, while in a formal onboarding, they are assisted in their new organizations by means of a written set of coordinated policies, procedures and actions. The levels of formality and comprehension of an onboarding program vary among companies. However, existing literature suggests that successful companies regard onboarding of new employees more formally rather than ad-hoc [11].
While there is a vast literature about onboarding in a diverse type of professions \cite{12, 82, 120, 136, 177}, there are only a few dedicated studies in software engineering research and, to the best of our knowledge, there is no holistic study that investigates how the onboarding of software developers is strategized.

To fill the existing gap, we carried out an empirical investigation on how the onboarding of software developers is strategized in three different companies. Our case companies are globally distributed and onboard developers to work in ongoing software product development efforts. We argue that such environments introduce additional challenges for both newcomers and software companies, since onboarding in globally distributed environments may happen both locally and remotely. Furthermore, newcomers may have to deal with large amounts of accumulated legacy code, which was written by the original developers who may not be working in the company any longer. To understand the peculiarities and challenges of onboarding processes in the mentioned environment, we share our findings from an exploratory multi-case study that is driven by the following research question:

- What are the functions (practices, tools, techniques, methods and technologies) employed by companies to onboard software developers in globally distributed projects?

The main contributions of this chapter are:

- A holistic analysis of the onboarding strategy employed to onboard software developers by three software companies (holistic, related to software engineering field, empirical);
- The use of an onboarding model to assist in the process of identifying areas of improvement in the onboarding process of software developers and software development teams (model applicability testing, theory-driven research);
- Recommendations that can be used by companies to improve their onboarding strategies with respect to local and remote developers.

The remainder of this chapter is organized as follows: Section 2 describes the background and related work. Section 3 presents the research design, followed by the results in Sections 4, Section 5 presents a discussion and implications for practice. Validity threats and limitations are discussed in Section 6. Section 7 contains the conclusions and our view on future work.

2 BACKGROUND AND RELATED WORK

In this section, we describe the concept of onboarding (organizational socialization), the model for successful onboarding by Bauer \cite{9} and summarize related work that focuses on onboarding of software developers.
2.1 Onboarding

To improve the effectiveness of the talent management systems beyond effective recruitment and new employee selection, companies shall consider the strategic use of onboarding. Onboarding refers to the mechanism through which newcomers acquire the necessary knowledge, skills, and behaviors to become effective organizational members and insiders [10, 203]. Research and conventional wisdom both suggest that employees get about 90 days to prove themselves in a new job [9].

Klein et al. [120] affirm that the research on onboarding can be divided into four distinct perspectives:

- Stages through which newcomers progress [33, 73].
- Actors involved with the onboarding of newcomers [6, 158].
- Tactics and practices employed by organizations for onboarding newcomers [9, 117, 203].
- Content to be learned by newcomers during the onboarding [42, 73].

Considering that the main focus of this paper is on onboarding tactics and practices, we elaborate further on this perspective, describing the main models of onboarding: Van Maanen and Shein’s model [203], Jones’ model [108] and Bauer’s model [9].

2.2 Van Maanen and Shein’s model

Van Maanen and Shein [203] proposed a theoretical explanation regarding role orientation in the context of onboarding. The model categorizes onboarding tactics in six dimensions:

- **Collective vs. individual** – Collective onboarding occurs when a group of newcomers go through onboarding activities and acquire experiences together (e.g., boot camps). Individual onboarding occurs when newcomers go through separate from other newcomers (e.g., apprenticeship).
- **Formal vs. informal** – Formal onboarding relates to tactics in which newcomers are segregated from other employees (e.g., policy academies). Informal onboarding relates to tactics that have no or little separation between newcomers and other employees (e.g., on-the-job training).
- **Sequential vs. random** – Sequential onboarding refers to the extent to which discrete steps regarding the onboarding phases are specified for the newcomers, while random onboarding tactics do not specify any sequence of steps.
- **Fixed vs. variable** – Fixed onboarding occurs when there is a timetable associated with each step of the onboarding process, so that a newcomer knows the exact time required to complete each step. Variable
onboarding does not associate any time with the onboarding steps. Rather, newcomers receive some clues regarding when they should consider an onboarding step as concluded.

- **Serial vs. disjunctive** – Serial onboarding takes place when experienced employees serve as models for newcomers (e.g. a new police officer works for an extended period with some veteran police officer). Disjunctive onboarding refers to the tactics wherein no guidelines or models are provided to newcomers.

- **Investiture vs. divestiture** – Investiture onboarding occurs when an organization prefers that newcomers keep their personal characteristics and make use of their own skills, values, and attitudes. Divestiture takes place when an organization rejects and removes the personal characteristics of newcomers.

According to this model, the way newcomers respond to their roles differs due to the onboarding tactics used by organizations. This means that organizations can support newcomers by giving relevant information in different ways.

### 2.3 Jones’ model

Jones’ model [108] was built upon Van Maanen and Shein’s Model [203] and reduces the original six dimensions to two:

- **Institutionalized** onboarding occurs when tactics are implemented in structured programs and newcomers receive formal group orientation and mentoring. This dimension is composed by the following dimension categories of Van Maanen and Shein’s Model: collective, formal, sequential, fixed, and serial investiture.

- **Individualized** onboarding takes place when newcomers start working from the beginning and must learn the norms, values, and expectations on-the-fly. This dimension is composed of the following dimension categories of Van Maanen and Shein’s Model: individual, informal, random, variable, disjunctive, and divestiture.

Institutionalized onboarding is related to formal tactics, while individualized onboarding is related to informal tactics. Companies considered as successful regarding the onboarding of newcomers have more formal onboarding programs (institutionalized onboarding) [13, 36, 117].

### 2.4 Bauer’s model

Bauer et al. conducted a series of studies that resulted in an empirically based onboarding model [9–13]. The model was conceptualized to support
the design of onboarding programs, capitalizing on the fact that institutionalized onboarding is more successful than individualized onboarding [13, 36, 117]. While related to Van Maanen and Shein’s model and Jones’ model, Bauer’s model has a finer grain level than the previous models; it aggregates practices, techniques, methods and technologies (functions) that are related to successful onboarding (Figure 6.1).

The benefits of this model are two-fold: i) it facilitates the evaluation of current state of onboarding programs in real projects, supporting the identification of areas to improve; ii) it provides a set of good practices that can be used by organizations to improve their onboarding programs. Considering the main goal of this paper, Bauer’s model was considered the most adequate model and, thus, used in our investigation.

According to Bauer, onboarding has four distinct levels, known as the Four Cs, which are the building blocks of successful onboarding [9]:

- **Compliance** is related to teaching employees basic legal and policy-related rules and regulations.
- **Clarification** is related to ensuring that newcomers understand their new jobs and the related expectations.
- **Culture** is related to providing newcomers with a sense of organizational norms, including both formal and informal.
- **Connection** is related to the interpersonal relationships and information networks that newcomers must establish.

The extent to which an organization focuses on each C determines its onboarding strategy. The combination of tools, practices, recommendations, performance goals and measurement milestones constitutes an onboarding strategy, which is often formalized in an onboarding plan [9].

The success of an onboarding strategy is related to short-term and long-term outcomes. Short-term outcomes are associated with the adjustment of new employees to their new jobs [9]. They go through a series of four adjustments:

- **Self-efficacy** is the first level of adjustment and represents the degree that new employees feel confident when carrying out the work related to their new jobs. The more self-efficacy, the more motivated and successful an employee has the potential to be [182]. Furthermore, self-efficacy is associated with high job satisfaction and low turnover [13].
- **Role clarity** is the second level of adjustment and represents how well new employees understand their role and expectations. Measures of role clarity are seemed as effective predictors of job satisfaction and organizational commitment and performance [74].
- **Social integration** is the third level of adjustment and represents the extent to which new employees feel socially comfortable and ac-
cepted by their colleagues and superiors [157]. Effective social integration is related to committed employees and low turnover rates [9].

- **Knowledge of culture** is the fourth level of adjustment and represents the possession of knowledge about the prevalent organizational culture (politics, goals, values and a company’s unique language) and the extent to which the new employees fit to it [9].

Long-term outcomes of onboarding are related to attitudes and behaviors. In the long-term, effective onboarding leads to: higher job satisfaction, organizational commitment, lower turnover, higher performance levels, career effectiveness and lowered stress [9, 138, 146]. Although very important, long-term outcomes are not covered in this chapter; our investigation encompasses the aforementioned short-term outcomes of onboarding.

The functions of Bauer’s model are grouped into six categories [9]:

- **Recruiting** – In many organizations, recruiting is not integrated with the onboarding plans and is treated as a separate function. However, existing literature shows that this integration (e.g. through realistic job previews or early involvement of stakeholders) gives to people being recruited a larger and more accurate amount of information about the company and job. As a result, this facilitates the adjustment of new employees, specially self-efficacy, role clarity and knowledge of culture [119].

- **Orientation** – Formal orientation programs help newcomers to understand important aspects of their jobs and organizations, as the company’s culture and values [118]. Moreover, they also help newcomers feel welcome by presenting them to other individuals within the organization. Computer-based orientation programs can help to keep consistency to the program in different locations. This function facilitates all four types of adjustment [9].

- **Training** – They are mandatory to give to the new employees the confidence, clarity and skills required by their job. New employees can receive training about hard skills and soft skills. The type of training depends on the self-efficacy of new employees in relation to what is demanded by the job. As a result, training facilitate the adjustment of new employees, specially self-efficacy, role clarity and knowledge of culture [9].

- **Coaching and support** – Mentors can teach newcomers about the company, provide advice and help with job instruction. Existing research shows that new employees with mentors acquire more knowledge about their new company than the ones without mentors [165]. Furthermore, mentoring programs and opportunities for informal interaction with colleagues certainly help the new employees to adapt
more easily to the new work environment. This function facilitates all four types of adjustment factors \[9\].

- **Support tools and processes** – Tools and formal processes are of great value for onboarding success. According to Bauer \[9\], there are three tools/processes that are related to successful onboarding: a written onboarding plan, which is a formal document that contains the timeline, goals, responsibilities and support available to each newcomer; stakeholder meetings, which occur in specific intervals, involve all the onboarding stakeholders, and allow newcomers to get the information they need; onboarding online, which can help to track the onboarding progress against development and career plans, and also help stakeholders to identify any additional help that new employees may need. This function facilitates all four types of adjustment factors \[9\].

- **Feedback** – Newcomers need constant feedback and guidance to understand and interpret the reactions of their co-workers. Feedback can be mainly provided in two different ways \[9\]: performance appraisals and 360-degree feedback, wherein the new employees are evaluated and receive developmental feedback and are also able to know how others view them; employee-initiated information and feedback seeking, wherein the new employees proactively seek feedback. This function facilitates the adjustment of new employees, especially self-efficacy, role clarity and knowledge of culture \[9\].

Figure 6.1: Model to support the development of onboarding strategies (adapted from Bauer \[9\]).
Some studies approach the onboarding topic in the context of software development (See Table 6.1), commonly addressing the functions of Bauer’s model in isolation.

In relation to recruiting, Tockey [197] has investigated the recruitment of software developers and has identified a misalignment between what hiring managers ask for and what they really need. By mapping postings for software developer jobs and issues related to software projects, they identified that the postings did not include skills that could be useful to address identified problems (e.g., code review skills to address software quality problems).

Regarding orientation, the importance of effective socialization (a part of the orientation process in the Bauer’s model) is emphasized by Casalnuovo et al. [41]. The study focused on the links between social relationships and onboarding. They explored GitHub to identify how the technical factors of past experiences and social factors of past connections to team members of a project impact their productivity. They identified that developers prefer to join projects in GitHub where they have pre-existing relationships. They also found that stronger social connections are related to lower initial and higher late productivity.

In relation to training, there is a vast literature that focus on the training function of onboarding in the context of software engineering. While it is beyond the scope of this chapter to cover comprehensively the literature related to training, and especially the hard technical skills, we would like to highlight a study related to soft skill training in the context of software engineering, which is key when onboarding newcomers according to Bauer [9]. Matturo et al. [142] conducted a field study to identify soft skills that are most important for product owners and Scrum masters. They identified that communication skills, customer orientation, and teamwork are the most important soft skills for product owners, while commitment, communication skills, interpersonal skills, planning skills, and teamwork are the most important soft skills for Scrum masters.

Regarding coaching and support, a number of studies focus on the role of mentoring for more effective onboarding. Falgerholm et al. [72] investigated the impact of mentoring on the onboarding of new developers in open source projects. They conducted an investigation that involved the onboarding of students from different universities in an open source project. Experienced developers mentored the students for three days. The activity of the mentored developers (students) was compared to the activity of non-mentored developers. They identified that the mentored developers were more active in the first 16 weeks of engagement than the non-mentored developers. In another study, Falgerholm et al. [71] investigated the influence of mentoring and project characteristics on the effectiveness and
efficiency of the onboarding process of software developers in open source projects. They used quantitative measurements of source code repositories, issue tracking systems, and developer’s discussions to identify how newcomers become contributing members of open source projects. They found that developers that receive deliberate onboarding support through mentoring were more active in the beginning. Britto et al. [27] investigated how software architects mentor new software developers and teams in a large-scale globally distributed legacy project. They identified that it is specially challenging to provide mentoring for teams located offshore, due to geographical and temporal distances. Furthermore, the existence of large amounts of complex legacy code amplifies the difficulty to mentor new remote development teams. Labuschagne and Holmes [128] investigated the effectiveness of mentored onboarding in open source projects supported by Mozilla Foundation. They compared two different onboarding approaches: one with focus on easy bug fixing without mentoring, and the other focused on mentored bug fixing (more complex bugs). It was found that the programs that implemented the onboarding strategies were not enough to automatically improve the chances that a developer becomes a long-term contributor.

Finally, some support tools and processes have been proposed to support the onboarding of developers. Monasor et al. [156] focused on developing a virtual environment to allow students to support soft skill training, which focus on acquire software communication skills in a more practical way. The virtual environment allows for the customization of different training scenarios, enabling instructors to adapt them according to different needs. Steinmacher at al. [192] created and evaluated a portal to support the onboarding of new developers in open source projects. The portal is based on a conceptual model of barriers [193] and the evaluation results suggest that it helped to alleviate barriers related to orientation and contribution process. Researchers have tried to facilitate mentoring through recommender systems. Cubranic at al. [53] and Malheiros et al. [139] developed tools that create a project memory automatically. The project memory can answer some questions made by newcomers, decreasing the need for human-based mentoring. In both cases, the tools were evaluated in the context of open source projects and were only successful to help newcomers to solve easy tasks. Canfora at al. [38] also developed a recommender system, but their focus was to allocate appropriate mentors for new developers in open source projects. They identified that the top committers are not necessarily the most appropriate mentors.

Three other studies worth mentioning in relation to our research question do not address any of the function categories directly but focus on barriers faced by new developers during the onboarding process. Steinmacher et al. [191] conducted a systematic literature review on the barriers faced by newcomers to contribute in open source projects. They identified
15 barriers that hinder the onboarding process of new developers in open source projects, five categories of barriers (social interaction, newcomers’ previous knowledge, finding a way to start, documentation, and technical hurdle) and three origins (newcomers, community, or product). Steinmacher and Gerosa [193] conducted a survey to investigate in more detail the challenges that new developers face to select the first task to start contributing in open source projects. They found that new developers are not enough confident to choose their initial task and, thus, need support from the open source community to select an appropriate task. In both studies, none of the functions of Bauer’s model are addressed directly, but they indicate the importance of implementing them to address the identified barriers/challenges. And finally, Smite and van Solingen [211] studied an outsourcing relationship between a Dutch customer and an Indian vendor, and found that it took longer for remote developers in India to climb up the learning curve in comparison with the developers onboarded on site in the Netherlands. This was primarily due to the distance to the sources of requirements and development knowledge, and the lack of local developers with experience. They also found that on site developers received more attention and training, while offshore developers were expected to be mentored and trained by the outsourcing vendor company.

As shown in Table 6.1, feedback is the only function category not covered by the related work, while coaching and support tools and processes are the most covered function categories. Furthermore, the coverage of onboarding function categories in the existing literature dedicated to software developers is fragmented (function categories are investigated in isolation). Finally, most of the studies we highlighted in this section focus only on open source projects. Therefore, to the best of our knowledge, no study reports a holistic investigation of software developer onboarding, accounting for all the function categories together in closed source globally distributed projects.

3 RESEARCH DESIGN

To address the research question formulated in this chapter, we conducted an exploratory holistic multi-case study [181]. We investigated three different cases, which were selected through convenience sampling as adequate cases to investigate strategies to onboard software developers/teams. All cases involve globally distributed legacy projects. In this section, we describe each case, unit of analysis, and the data collection, preparation and analysis processes.
Table 6.1: Relationship between the related work and Bauer’s model.

<table>
<thead>
<tr>
<th>Study</th>
<th>Focus</th>
<th>Category</th>
<th>Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>[197]</td>
<td>Recruiting</td>
<td>Recruiting</td>
<td>Unclear</td>
</tr>
<tr>
<td>[41]</td>
<td>Onboarding social factors</td>
<td>Orientation</td>
<td>Open source</td>
</tr>
<tr>
<td>[142]</td>
<td>Soft skill training</td>
<td>Training</td>
<td>Open source</td>
</tr>
<tr>
<td>[72]</td>
<td>Mentoring</td>
<td>Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[71]</td>
<td>Mentoring</td>
<td>Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[27]</td>
<td>Mentoring</td>
<td>Coaching and support</td>
<td>Closed source</td>
</tr>
<tr>
<td>[128]</td>
<td>Mentoring</td>
<td>Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[156]</td>
<td>Soft skill training tool</td>
<td>Support tools and processes; Orientation</td>
<td>Academia</td>
</tr>
<tr>
<td>[192]</td>
<td>Recommender system to support mentoring</td>
<td>Support tools and processes; Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[53]</td>
<td>Recommender system to support mentoring</td>
<td>Support tools and processes; Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[139]</td>
<td>Recommender system to support mentoring</td>
<td>Support tools and processes; Coaching and support</td>
<td>Open source</td>
</tr>
<tr>
<td>[38]</td>
<td>Recommender system to select mentors</td>
<td>Support tools and processes; Coaching and support</td>
<td>Open source</td>
</tr>
</tbody>
</table>
3.1 Case description

**Case 1** is a Polish company that provides services in dedicated IT solutions and applications, IT outsourcing, IT consulting, customizations and training. The main market segments of this company are telecommunications M2M (Machine to Machine), healthcare ERP (Enterprise Resource Planning), business intelligence applications, and finances and banking solutions. The company has subsidiaries in Sweden, Ukraine, and Belarus. The investigation focused on people involved in the development of telecommunication solutions. The people we interviewed work in a project in which the company provides software development services to another company with strong presence in the telecommunication market segment.

**Case 2** is a leading supplier of intelligent transportation systems to the public transport sector, including fare collection, travel information, infotainment, fleet management and traffic management. The investigated two teams consisting of nine members each including one tester in each team (in Poland), product owners are in Norway and customers mostly in Scandinavia. Software development teams follow a Scrum-based process, where a release is about 6 months long. The two teams have a complex environment of products and the domain knowledge is not easy to acquire. A large part of the systems consists of legacy code, which was produced by developers that are not anymore available. Some people are recruited to work on the new part of the system and some to work on the legacy part of the system only.

**Case 3** is a large-scale distributed project associated with the development and evolution of a large software product in Ericsson, a large Swedish company that commercializes telecommunications-related products. The product originated in Sweden and has evolved for almost 20 years, and, by the time of this investigation, involved 188 employees (15 software architects, 134 developers working in 19 formal teams) distributed across Sweden (five software development teams), India (10 software development teams), Italy (one software development team), USA (one software development team) and Poland (two software development teams).

3.2 Data collection

In all three cases, we collected data through semi-structured individual and group interviews, and a workshop. Most of the interviews were conducted face-to-face. We recorded the audio and made notes during the conduction of the interviews/workshop. More details about the interviews are presented in Table 6.2.

To collect the data in **Case 1**, we conducted six individual interviews and two group interviews (90 min long). We interviewed the three prod-
uct managers who are highly involved in the onboarding of new developers. We also interviewed the manager of the technical training center, two developers and two teams of developers to have different views on the onboarding process of Case 1 (60 min long). All the interviewees are from one of the sites located in Poland. The interviews were conducted face-to-face in Poland in May 2016.

To collect the data in Case 2, we conducted four group interviews, in which we interviewed two teams (45 min long), and four senior developers highly involved with the onboarding of developers (two group interviews, each 40 min long). All the interviews were conducted face-to-face in Poland in November 2016.

Finally, to collect the data regarding Case 3, we conducted three individual interviews and one group interview. We interviewed the product manager, who is highly involved in the onboarding of new developers. We also interviewed a software architect (located in Sweden), a senior developer (located in the USA) and conducted a workshop with a team of developers (located in India) to have different views on the onboarding process of Case 3. The data was collected from October 2016 to January 2017. All the interviews (60 min long) and the workshop (80 min long) were conducted in Sweden face-to-face, except for one that involved a developer located in the USA, which was carried out via Skype.

Through the interviews, we were able to identify how the onboarding functions are implemented in each case’s strategy (if so). Furthermore, we could identify aspects that can be improved in each respective onboarding strategy.

To conduct the interviews, we developed two semi-structured interview guides, one to interview managers (interview guide 1 in Appendix A) and the other to interview developers (interview guide 2 in Appendix A). The interview guide 2 was also used to conduct the workshop in Case 1.

3.3 Data preparation and analysis

Before analyzing the collected data, we transcribed all interviews related to Case 1 and Case 2. Then, we asked the interviewees to check the transcriptions. Two interviewees from Case 1 identified issues in their respective transcriptions, which were fixed. Regarding Case 3, we did not record the audio of the interviews and the workshop to make the participants more comfortable and prone to speak out. In all interviews, we took notes to facilitate posterior analysis. The notes included key points related to the questions that were posed during the interviews. The notes were discussed with the respective interviewees, to ensure that they reflected what was discussed during the interviews. In the workshop, we asked the participants to provide information about the challenges they faced during their on-
Table 6.2: Details about the interviews.

<table>
<thead>
<tr>
<th>Case</th>
<th>Date</th>
<th>Role</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11/05/2016</td>
<td>Software developer</td>
<td>2 years</td>
</tr>
<tr>
<td>1</td>
<td>11/05/2016</td>
<td>Technical training center manager</td>
<td>5 years</td>
</tr>
<tr>
<td>1</td>
<td>11/05/2016</td>
<td>Product manager</td>
<td>12 years</td>
</tr>
<tr>
<td>1</td>
<td>11/05/2016</td>
<td>Product manager</td>
<td>10 years</td>
</tr>
<tr>
<td>1</td>
<td>11/05/2016</td>
<td>Software developer</td>
<td>8 years</td>
</tr>
<tr>
<td>1</td>
<td>12/05/2016</td>
<td>6 Software developers</td>
<td>2 with less than 6 months and 4 with more than 3 years</td>
</tr>
<tr>
<td>1</td>
<td>12/05/2016</td>
<td>6 Software developers</td>
<td>2 with more than 2 years and 4 with less than 1.5 years</td>
</tr>
<tr>
<td>2</td>
<td>20/11/2016</td>
<td>2 Software developers</td>
<td>Less than 3 years</td>
</tr>
<tr>
<td>2</td>
<td>20/11/2016</td>
<td>2 Software developers</td>
<td>Less than 3 years</td>
</tr>
<tr>
<td>2</td>
<td>20/11/2016</td>
<td>2 Software developers</td>
<td>More than 5 years</td>
</tr>
<tr>
<td>2</td>
<td>20/11/2016</td>
<td>2 Software developers</td>
<td>More than 5 years</td>
</tr>
<tr>
<td>3</td>
<td>17/01/2017</td>
<td>Product manager</td>
<td>16 years</td>
</tr>
<tr>
<td>3</td>
<td>03/10/2016</td>
<td>Software architect</td>
<td>10 years</td>
</tr>
<tr>
<td>3</td>
<td>10/10/2016</td>
<td>Design lead</td>
<td>15 years</td>
</tr>
<tr>
<td>3</td>
<td>21/12/2016</td>
<td>5 Software developers</td>
<td>More than 2 years</td>
</tr>
</tbody>
</table>
boarding, with a special focus on their learning process. The developers were asked to write down, independently of each other, the challenges that, in their opinion, impacted their learning processes. After 10 minutes, individual opinions were discussed within the group. The results were aggregated in a report that was verified by the participants.

To analyze the collected data, we followed the coding process described by Robison and McCartan (open coding) [176]. We used the function categories described in Bauer’s model as primary codes. We coded the interview transcriptions, notes and workshop report using the defined codes, aiming at identifying how onboarding was strategized (implemented functions) in each case. Then, we determined the order and duration of the implemented functions in each case. We designed diagrams to visualize the functions (see Figures 6.2 to 6.5), emphasizing the functions dedicated to the legacy (functions in grey color) and distinguishing between the functions planned and implemented by offshore sites locally and functions planned and implemented centrally by the main sites. The coverage of the onboarding functions, as well as similarities and differences between the cases, were then summarized (see Tables 6.3 to 6.8) and discussed.

Finally, we derived recommendations for software companies onboarding developers and teams in globally distributed legacy projects (see Section 5). To do so, we paid special attention to functions that were implemented in the investigated cases to address the challenges of onboarding on a distance, or dealing with legacy code. Some of the practices drive the readers’ attention to the challenging areas and warn about the necessity to have a proactive action plan, while others propose concrete action. The recommendations resulted from the combination of practices well-evaluated by the interviewees and our observations from the three cases.

4 Results

In this section, we present the results associated with each case. In all cases, the companies employed semi-formalized onboarding strategies. Tables 6.3 to 6.8 summarizes the results, wherein green color represents fully implemented functions, yellow represents partially implemented functions, and red represents not implemented functions (see Section 2 for more details about the onboarding functions and categories). The results are further elaborated in the remainder of the section.

4.1 Case 1

The summary of the onboarding functions implemented in Case 1 is given in Figure 6.2. We learned that in Case 1 recruitment is integrated with the existing onboarding strategy. The company organizes summer schools for
Table 6.3: Results regarding recruitment.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recruitment integrated with onboarding</td>
<td>Fully Integrated</td>
<td>Partially integrated</td>
<td>Partially integrated</td>
</tr>
<tr>
<td>Realistic job previews for newcomers</td>
<td>Summer schools</td>
<td>Not implemented</td>
<td>Not implemented</td>
</tr>
<tr>
<td>Stakeholder involvement in recruitment</td>
<td>Senior developers participate in technical interviews</td>
<td>Senior developers in already established sites participate in technical interviews and CV screening</td>
<td>Senior developers in already established sites participate in technical interviews and CV screening</td>
</tr>
</tbody>
</table>

Table 6.4: Results regarding orientation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal orientation course or material for the newcomers</td>
<td>Not implemented</td>
<td>Not implemented</td>
<td>Not implemented</td>
</tr>
<tr>
<td>First day at the job is special</td>
<td>The whole first week at the new job is dedicated to familiarization with the environment</td>
<td>The first month is dedicated to learning on the job.</td>
<td>The whole first week at the new job is dedicated to familiarization with the environment</td>
</tr>
</tbody>
</table>
Table 6.5: Results regarding coaching and support.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mentoring programs</td>
<td>A mentor assigned to new developers inside a boot camp. Afterwards, a mentor assigned to a new developer or a mentor team assigned to a team of new developers</td>
<td>One or several mentors assigned to new developers in a team.</td>
<td>A mentor assigned to a new developer or a group of new developers</td>
</tr>
</tbody>
</table>

Table 6.6: Results regarding training.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal training on hard skills and/or soft skills</td>
<td>3 months long training focusing on technical and methodological knowledge, provided in boot camps</td>
<td>Not implemented</td>
<td>3 months long training focusing on technical, methodological, and product knowledge</td>
</tr>
</tbody>
</table>
Table 6.7: Results regarding support tools and processes.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Onboarding plans</td>
<td>An intranet with useful company material</td>
<td>An intranet with useful company material and an instruction page</td>
<td>Onboarding plans, an intranet with company material</td>
</tr>
<tr>
<td>Regular stakeholder meetings</td>
<td>Face-to-face meetings</td>
<td>Face-to-face meetings</td>
<td>Face-to-face and videoconferencing</td>
</tr>
<tr>
<td>Own progress monitoring</td>
<td>Not implemented</td>
<td>Not implemented</td>
<td>Progression spreadsheets</td>
</tr>
</tbody>
</table>

Table 6.8: Results regarding feedback.

<table>
<thead>
<tr>
<th>Function</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance appraisals</td>
<td>Face-to-face during meetings with mentors and immediate managers</td>
<td>Face-to-face during appraisal meetings with immediate managers</td>
<td>Face-to-face during meetings with mentors and immediate managers</td>
</tr>
<tr>
<td>360-degree feedback</td>
<td>Feedback from mentors via code reviews and face-to-face</td>
<td>Feedback from mentors via code reviews</td>
<td>Feedback from mentors via code reviews</td>
</tr>
</tbody>
</table>
newly graduated bachelors from the local universities, during which the young professionals receive technical training within the actual software teams. This provides the candidates with realistic job previews. Successful attendees of the summer school receive a job offer. In this case, a series of job interviews may be carried out. Senior developers participate in the evaluation of candidate developers during the technical interviews. Selected candidates proceed to a special onboarding program called “boot camp” and receive an initial 3-month contract, which can be extended if the new developer performs well and shows good potential. Other members of the team, where a new developer is allocated, conduct the evaluation of performance and potential.

A boot camp is a 3-month long program in which small groups of newcomers have to carry out a small project, usually involving a toy task for learning purposes and to be able to evaluate the potential of the candidate. Boot camp aims at learning the intended programming languages, tools, environments, and ways of working, and facilitates socialization and team building. During a boot camp, a senior developer is assigned to support a group of developers and is responsible for facilitating the required training and provide knowledge of corporate behavior. Therefore, it is fair to say that boot camps serve as a premise for orientation, training and coaching.

There is no formal orientation of the new developers. The welcoming of the newcomers in a boot camp is done in an informal way is conditional in its nature, since the more permanent recruitment of developers happens upon the successful completion of the boot camp, three months later.

Coaching and support, as noted earlier, is a central part of a boot camp program. The mentors assigned to the developers are used as the first-hand contacts for any help needed. After a boot camp, the new developers receive additional support, but the way it is provided by the company differs depending on the current circumstances. If few developers are onboarded, each one of them will be assigned to a mentor (senior developer) within the team. If many developers are onboarded together, often a new team is created with the new developers. The new team is assigned to a mentor team, which is a mature software development team. In general, the new
team works on the backlog of the mentor team, “shadowing” the work of the mentor team in the beginning and doing small tasks more independently afterwards. This allows an experienced team to take off some workload while helping the new team, without losing overall productivity. Further support is provided through communities of practice, wherein new developers can interact with senior developers to acquire technical and methodological guidance. It is important to note that socialization and seek for help from others is emphasized in the company and noted by the new recruits.

Although the company employs a learning-by-doing approach, i.e. there is no big emphasis on traditional classroom training, the company facilitates this through formal training programs provided during the summer schools and the boot camps. There are two types of training: external training, which are focused on technical (e.g. domain, programming languages) and methodological (e.g. agile practices) training, and are provided to all units by the corporate training center; and internal training, which are focused on product knowledge and are provided by a unit for its developers/teams. After receiving the formal training, the new hires are further incorporated in the teams to “shadow” the more experienced developers in their daily tasks. This is seen as an important and relatively secure way of acquiring the product knowledge and the knowledge of the legacy code.

The company also provides soft skill trainings (e.g. multi-cultural training), but they are only offered at the later stages of the developers’ careers.

The company provides some support tools and processes, including an intranet and wiki pages, wherein documents about the ways of working and the product are available.

When it comes to feedback, the company facilitates two types of feedback. The candidates have a possibility to provide feedback on the onboarding process by requiring additional training within a boot camp or after being incorporated in a team. The candidates also receive feedback from the mentors face-to-face or via code reviews. Mentors and immediate managers also evaluate and provide feedback about the performance of newcomers during performance appraisal meetings.

4.2 Case 2

The summary of the onboarding functions implemented in Case 2 is given in Figure 6.3. We learned that recruitment in Case 2 is partly integrated with the onboarding process. Although no formal realistic job preview is provided, senior developers are included in the process of recruitment, both during the CV (curriculum vitae) screening and when conducting technical interviews. The two main factors of success mentioned during the interviews were related to a good technical meeting and knowing whom
they will work with if employed. These factors determined whether the recruitment process was good or not. Also the existing developers perceived that it is important to be a part of the recruitment decisions. Recruited developers commented that the technical interview allowed them to know more about the work in the company.

The orientation of new developers is done in an informal way, i.e. there is no formal orientation program. In general, the first week at the new job is dedicated to familiarizing with the new environment, further the new recruits are given one month to learn their way, but besides formal introduction to the team they will be working with, this process is not formalized. One interviewee commented that he would like to have more information about the company structure, and who is responsible for each department. Otherwise those new to the company must ask different people about who is responsible for what, knowing who is who is especially hard in a globally distributed company. Another interviewee also commented that the goals of the project must be clear since the beginning.

The company provides coaching and support for the new developers, but the way it is done depends on the amount of people being onboarded. In each team, there is someone that does the mentoring for the new employees, depending on which part of the system the new employee will work on. Some projects are very complex with respect to domain-specific knowledge, so there is nobody who is knowledgeable in all aspects. There is one month learning time given by the company to a new employee, but there is no clearly separated time for the mentors to spend time with the new employees. It is informally known that the mentors’ productivity will slow down, but they do not have any dedicated time for mentoring explicitly. At the same time, they do activities such as explaining the domain, showing how to setup the environment, doing code reviews and answering the questions as they arise. Work as a mentor puts additional pressure on the mentors and other experienced team members, because they must keep up with their own productivity and at the same time take care of the new employees. The best mentoring strategy happens when there is an overlap of the time when the senior developer is leaving the company and the new
one is arriving, so the new developer can receive the introduction from the
person that he/she is replacing. Some new developers thought that one
month was too much time for just learning, and after two weeks they have
started to take some simple tasks to solve. The company does not provide
any formal training for the new employees.

Regarding support tools and processes, the company has an intranet wherein
documentation is made available, technical sessions about architecture of
the system and documentation (“how to” navigate on the wiki pages, con-
fluence etc.), and an instruction page (how to set up environments, virtual
machines, where to find information needed, who has specific knowledge
and acronyms and glossary of terms). Not all information is equally struc-
tured and it may be hard for new developers to find the information they
need.

Feedback is mainly used to identify whether a new developer needs more
support or identify whether it is worthwhile to hire a new developer per-
manently. Appraisal meeting with all employees are organized every six
months, wherein the employees receive feedback on their performance and
have an opportunity to speak about their own experiences in the company.
The team members also use code reviews to support the onboarding pro-
cess of new developers.

4.3 Case 3

In Case 3, we explored the challenges associated with remote onboarding,
which at the same time was carried out in the context of a complex legacy
product. Given the changing market demands, developers and teams were
frequently onboarded in the project we have studied in Case 3. This hap-
pened both locally in Sweden and remotely in USA, Italy, China, Turkey,
Poland and India. During the 20 years of product existence, over 30 teams
were onboarded and more than half removed from the project, not to men-
tion individual onboarding. Given the distributed nature of the project
some onboarding functions were organized by the central project stake-
holders in Sweden, while others occurred remotely and were handled by
the local management. The processes for onboarding differed for sites with
considerable experience and already established teams from those sites,
which just started their engagement in the project. The summary of the
two onboarding strategies employed in Case 3 are given in Figure 6.4 and
6.5 respectively.

The recruitment of the new employees is organized by each site indi-
vidually, while the demanded skill profiles are formulated by the central
location. The company does not provide realistic job previews, however,
senior developers participate in the recruitment process and the evalua-
tion of new candidate developers during the technical interviews in the
already established sites local. In general, inexperienced developers have an initial 6-months contract and experienced developers have an initial 1-month contract, which can be extended depending on the performance of the new employees, which is evaluated during the time of the first contract. Senior developers, commonly those from the central location, evaluate new developers by reviewing their work outcomes, i.e. the code. Overall, we conclude that the recruitment is partially integrated with the onboarding process.

The orientation of new developers is carried out in an informal way, i.e. there is no formal orientation program, and is up to the local stakeholders define how to implement orientation. In general, the new recruits are given one week at the new job to familiarize with the new environment, get to know the key people and co-workers (socialization), and acquire the basics about the existing ways of working. The way it is done depends on the amount of people being onboarded; if few people are onboarded, the orientation is carried out on an individual level; and if many developers are onboarded at the same time, the company provides a group level orientation. In both cases, a senior local developer or a manager provides an informal walk-around and discussion-based orientation.

The company provides three months long formal training for the new employees, when many developers are recruited. The training program fo-
cused mainly on product knowledge and required technical and method-
ological skills. The company also provides training related to soft skills, e.g.
a cross-cultural communication course for those working in a distributed
way, but they are not part of the onboarding strategy. Rather, they are pro-
vided in a later stage of an employee’s career. If just one person is to be
onboarded, no formal training is provided.

The company employs a learning-by-doing approach and new develop-
ers soon start to work with real tasks under careful coaching and support. De-
pending on the amount of people being onboarded, the company assigns
mentors on an individual or a group level. The mentors are responsible
for ensuring that the new employee or employees do not cause problems
that impact the whole product. Therefore, the new hires often start by
developing test cases for the product regression testing framework under
careful supervision, and then move on to more challenging tasks. Notably,
the onboarding process differs for the already existing sites and the newly
established sites. When many developers are onboarded in an already es-
tablished site, the common practice is to group them and allocate one or
two senior developers into the new team. In some cases, these groups are
separated after two months, the new developers are integrated in existing
teams, and the senior developers return to their original teams. The
special situation arises when the new developers are onboarded in remote
sites with no senior developers present for local coaching and support. It
is often hard to relocate sufficient number of senior developers from one
country to another for a long period of time. To address this, the company
first brings the new developers from the remote location to Sweden for
both training and initial period of supervised work (four to six months in
total). Then a senior developer follows the developers to the remote loca-
tion to provide on site mentoring for the next four to six months. Finally,
the new developers continue receiving support on a distance.

Regarding support tools and processes, the company uses many resources
to make the onboarding of new developers successful. All tools and pro-
cesses are centralized. For the remote teams, there is a formal onboarding
plan, with the goals, milestones and training associated with the new de-
velopers. As for the other functions, the goals are defined on an individual
or group level depending on the amount of people being onboarded at
the same time. For each developer, regardless location, there is an Excel
spreadsheet used to track their progression regarding the competence they
must acquire. This spreadsheet also contains the main source of knowl-
edge they can use to acquire the required competence. This file must be
updated to a system, which is used by immediate managers and mentors
to also follow the progression of new developers. Another tool is the cor-
porate intranet, wherein documents about the ways of working and the
product are available, and which is maintained centrally.
The new developers receive continuous feedback on their work outcomes (i.e. code) through code reviews. Local senior developers (if any) and software architects from the central location not only evaluate the performance, but also transfer product knowledge to support the new developers. Based on the received feedback, new developers may require more formal training. At the same time, the status of performance is used by the immediate managers locally to identify whether a new developer needs more support. Such checks are performed together with the mentors on a weekly basis. Code reviews also serve as a track record used in permanent employment considerations. Local stakeholders decide whether to hire or not permanently a new developer.

5 DISCUSSION

In this chapter, we have used the Bauer’s model for successful onboarding to analyze the onboarding functions and strategies in three software companies diverse in domains and size. In the following, we first discuss the degree to which each organization has formalized and structured their onboarding strategies. Then, we discuss the new challenges for onboarding developers to work with legacy systems and when onboarding remotely. Based on the results, we bring forward a list of recommendations that can support companies to improve their onboarding strategies, as well as some implications for future research on this topic.

5.1 Formality level of onboarding strategies

As noted earlier, companies may employ formal or informal onboarding processes [133, 134, 225], which depends on whether new employees learn about their new job on their own or following coordinated policies, procedures, and actions set by the company. Bauer [9, 13] and other researchers [36, 117] suggests that successful companies treat onboarding of new employees more formally, which can be done by specifically addressing the four essential components of onboarding – Compliance, Clarification, Culture, and Connection [9]. In other words, to make the onboarding successful, companies are expected to explicitly support newcomer familiarization with the legal policies and regulations, job-related training, organizational norms, and networking and building interpersonal connections with other employees important for completion of the job tasks.

In our investigation, we have specifically focused on the clarification and the connection building blocks in Bauer’s model. Unfortunately, we did not study the familiarization with the legal policies, regulations, and the corporate culture because it requires much more observation and an ethnographic approach.
In relation to Connection, the case companies employed a few practices that primarily included daily work practices for all employees and were not formalized as a part of the onboarding practices. For example, walkthrough-the-office type of introductions of the newcomers, exchange visits, and other contact building activities are used as a part of the orientation, but are not institutionalized. Participation in communities of practice, and team events were other sources of new contact acquisition. The more formalized functions based on the three studied cases were related to the coaching and support. In Case 1, for example, the company integrated the newcomers in existing teams to foster interpersonal connections. To the best of our knowledge, related literature in software engineering does not focus on this component.

With respect to clarification, we found that in all three case companies the most formally treated activities were those related to ensuring that newcomers understand their new job. Related literature [197] suggests that companies shall also clarify their expectations and provide candidates with detailed information about a particular job, to enable them to better decide whether the offered position matches their aspirations. However, we found that among the studied companies only one (Case 1) provided realistic job previews, but in Case 2, the team members participate in the interviews and can answer the questions of the newcomers and all get to know who they will be working with. After recruiting the new developers, all three companies put a strong emphasis on further anchoring the understanding of the job routines through a formally established mentoring program, as also suggested by related literature [71, 72]. In fact, in two of the cases (Cases 2 and 3), assignment of mentors was the prime onboarding function. The actual job clarification and feedback was enabled through less formal process employed by the mentors and other employees in general. Finally, the three companies employed code reviews (over-the-shoulder, email pass-around, pair programming or tool-assisted code review) to enforce clarification, which is also suggested as a good practice by existing literature [47].

Interestingly, we found that depending on the number of people being onboarded, companies employed different levels of formality to onboard software developers, which we have not come across in the related literature. This impacted the offering of formal training programs, the number of mentors, the allocation of new developers into existing teams or formation of new teams, as well as the duration of training, coaching, and support. A valid question for future investigation is whether this means that large groups of new developers (onboarded in a more formal way) have a better onboarding results, or individual treatment and less formal strategy result in a better onboarding outcome. Unfortunately, we are not unable to answer this question in this paper.
In all three investigated cases, the onboarding processes were additionally challenged because of the considerable amount of legacy code that the newly employed developers had to learn. To support the newcomers, all companies provided formal training about the product knowledge and different ways to coach and support them on the job. Although we know from existing literature that mentoring is a common practice for onboarding software developers \[71, 72\], we observed in our case studies that in legacy projects, coaching and support may be required for a significantly longer period. We also found that this may negatively impact the mentors, as our interviewees complained about the mentoring role being a heavy slog. It affects the mentors’ productivity since they must stop their work to respond to the new employees’ requests and spend time on task switching due to disturbances. In Case 3, besides the main retainers of knowledge being in another country, much of the legacy code was written by people that no longer worked in the company, demanding even longer periods of intense mentoring, often provided offsite.

Our findings also suggest that the largest challenge for companies is to onboard remote developers to an ongoing product development, especially if it follows agile methodologies. Existing literature shows that it may be hard for developers to start being productive when they are onboarded remotely \[191, 192\]. In all three companies, agile ways of working meant that there is generally a higher emphasis on networking with colleagues rather than documenting the progress, work outcomes and the ways of working. Even though all three companies have put effort into making the general guidelines and product descriptions available through the corporate intranet and wiki pages, the maintenance of consistent documentation with high coverage was a challenge. The interviewees from Cases 2 and 3 reported that parts of the products they worked with were not up to date or insufficiently detailed. This meant that new developers were required to keep a continuous dialog with the mentors, which especially in Case 3, was challenged by the temporal and geographic distance.

Networking was also problematic because of the lack of cross-site contacts. Some of the interviewees stated that they do not know in person some of the developers with whom they need to collaborate. Existing literature shows that stronger social relationships are related to higher productivity \[41\]. Thus, to facilitate more personal contacts, the studied companies have invested into video conferencing facilities, frequent visits from headquarters to the remote sites, and exchange between developers from different sites.

Another issue we identified is related to onboarding strategy fragmentation (related literature does not focus on such an aspect). In projects with multiple sites, it may be impossible to implement all onboarding functions
in all sites in the same way. For example, in Case 3 the recruiting function was planned and performed locally, which meant that different processes and criteria were employed to hire developers in each site. It is fair to assume that different ways of planning and implementing the onboarding functions may lead to uneven onboarding results and that a company can be successful in one site but fail in another due to the local differences. Although we have not investigated the onboarding success or the process differences in detail, the very fact that companies do not always have control over parts of the onboarding processes is an interesting finding.

5.3 Recommendations and implications for practice

In this section, we outline the recommendations based on the cross-company analysis of the onboarding functions and the lessons learned by each of the studied companies, followed by a few general implications that we derive from the execution of our investigation.

For the companies that onboard developers in distributed legacy projects we advise:

• **During recruitment explain the expectations for the new hires.** As one of the interviewees from Case 2 stressed:

  “The technical interview made a difference on choosing to work here because I got to know what potentially I would do, what technologies I would be exposed, and how do they work here, and some of technologies and the way they work, like that they follow Scrum.”

  Among our cases this was the only company providing realistic job previews.

• **As a part of orientation in distributed projects, acknowledge the importance of formalizing and mirroring the onboarding program across sites.** Make the objectives, timelines, roles and responsibilities clear. Revise what a new employee needs to learn and summarize it in a written onboarding plan. Write up a list of orientation activities in a guide for the mentors that perform the orientation and/or the hand-out material for the newcomers.

• **Provide transparency into the project organization and key roles across sites.** As one of the offshore interviewees from Case 1 revealed:

  “One thing that comes to my mind is that our project is managed basically not here. It is managed abroad. Especially at the beginning, it was
very difficult to get to understand, how exactly is it managed and what is required. What aside from basic things that we must develop code and deliver somehow, what else is required of us in terms of process and testing, and filling forms and documents. And some meetings that take place in the other site, for instance, perhaps contained some important information that we missed, because we didn’t know that some meetings did take place and some information is just passed through unofficial channels and we miss out on that.”

• **Invest in traveling.** While walk-through-the-office is a common onboarding practice when performing orientation of the newcomers on site, developers onboarded in distributed projects shall receive a special kind of orientation to support the establishment of ties with the remote colleagues. One of the interviewees from Case 1 said:

“It is very important to travel to other sites to meet face-to-face people I work. It helps to make you more aware of what is happening, where to pay attention, and who to ask if something is unclear. Before, I was aware of people who can help me and had knowledge I need, but it was hard to ask someone, who I didn’t really know. After I visited the other site, it became much easier, it’s almost like going to other room in your workplace. It is a lot easier to ask and it is also easier for the person who you ask help from.”

• **Provide extensive coaching and support in legacy projects.** Remember that familiarization with legacy code requires hands-on training, as an interviewee from Case 1 revealed:

“Bootcamp is a good thing […], but there is a huge amount of knowledge to gain […]. When you start using the things you are learning, working for a while, you remember what you have learned and then everything becomes clearer”.

Our studies indicate that the more legacy a product contains, the longer the mentoring period. Therefore, companies shall prepare to prolong the mentoring, if needed. At the same time, it is important to include coaching in the individual plans for the mentors to reduce their stress.

• **Tailor the training program to suit the developers’ needs.** In Case 3, we learned that the provided onboarding activities can potentially clash with the expectations from the new hires offshore. As an interviewee from Case 3 explained:

“Here we expect our developers to learn by doing. So, as soon as possi-
ble, we involve them in real tasks. However, people are different and some developers ask for more training than others. In my opinion, this is also related to cultural differences.”

• Use tools to provide feedback. In all three case companies, code reviews were used to provide daily feedback on the work outcomes of the new hires. This was appreciated by the newcomers, as one of the recently onboarded developers from Case 2 mentioned:

“We have the process of code review, so we have discussions and comments on our code, and then testing. So, it is a good way to get feedback.”

At the same time, the companies shall also acknowledge the limitations of tool support. The case companies emphasized the importance of collocated mentoring in distributed projects. As one interviewee from Case 3 stressed:

“We help the developers located in India mainly via code reviews and Skype. These approaches work, but it is much easier to help them when they are here in Sweden.”

We would like to emphasize the importance of treating the onboarding as a multifunctional process. Our cross-case analysis indicates that the coverage and sophistication of the onboarding programs may differ based on the organizational contexts (e.g., the number of newcomers, the amount of legacy code, and the distribution across multiple locations). It seems difficult to come up with one general onboarding strategy that would be useful in all circumstances. For example, a full-fledged onboarding program, as proposed by Bauer, might be economically unfeasible for onboarding a single developer. Rather, the incorporation of a developer into a well-functioning team and their mentoring and support could suffice in this scenario. Nevertheless, a similar lightweight onboarding program may be insufficient for onboarding many new developers in a large legacy project. Therefore, our recommendation is to tailor onboarding strategies for the onboarding undertaking at hand.

6 Threats to validity

In the following, we discuss the validity threats associated with reliability, internal, construct and external validity described by Runeson and Höst [181].

Reliability is related to the repeatability of a study, i.e. how dependent are the data and analysis on the involved researchers [181]. To minimize
this threat, three researchers were involved in the design and execution of this multi-case study. Furthermore, we developed an explicit case study protocol to guide the investigation as suggested by [181]. Finally, the observations and findings were verified with the companies’ representatives to avoid false interpretations and inconsistencies. Besides, the researchers have a long-term relationship with the case companies, which means that the researchers know more about the context of the companies, not basing the findings and interpretations only on the focused interviews. However, the collected data is qualitative and is highly dependent on the involved interviewees. We mitigated this factor by using a known theoretical model for onboarding that is used in other areas.

**Internal** validity is related to factors that researchers are unaware of or cannot control regarding their effect on the variables under investigation [181]. The main internal validity threats related to this paper are investigator bias and interviewee bias. Investigator bias was mitigated by involving three researchers during the design of the interview and workshop guides (investigator triangulation). To mitigate interviewee bias, we interviewed people with different roles (data triangulation).

**Construct** validity reflects how well the measures used actually represent the constructs the study intends to measure [181]. The main threat to construct validity in our investigation is that we used only one method to measure each construct. Data triangulation (interviewing multiple people) partially addresses this threat and strengthens the produced evidence. Moreover, we conducted a sanity check together with company representatives to validate the collected data.

**External** validity is concerned with the generalization of the findings [181]. Since the main research method employed in this work is the case study method, the main findings are strongly bound by the context of the selected cases. To mitigate this threat, we conducted three case studies in three different companies that provide services and products in different domains. Furthermore, our main contribution lies in the cross-company comparisons based on which we concluded about the variability of the onboarding practices. As such, it shall be valid despite the limitations of the cases. The main contributions of this paper may be of interest and applicable to researchers and practitioners that work in similar contexts. To allow the transferability of the findings of this work, we detailed the description of the investigated cases, within the limits imposed by the associated non-disclosure agreements. Note that more details were provided regarding the Case 3 since we could describe the case in such a level of detail. We had to omit the name of the companies of Cases 1 and 2 due to the non-disclosure agreements we signed.
In this paper, we investigated the strategies employed by three different companies to onboard software developers in globally distributed legacy projects. To do so, we used the model for successful onboarding proposed by Bauer [9] and evaluated the coverage of the onboarding functions in all three cases.

In response to our research question, we learned that the employed onboarding strategies are semi-formalized and vary from company to company, and depend on the context of onboarding. When it comes to the specifics of the context of the chosen companies, i.e. the distribution and the legacy products, we found that these aspects significantly challenged the onboarding of software developers. We identified that in globally distributed projects some parts of the employed onboarding strategies differ between different sites. This means that some of the onboarding functions are centralized, while others are executed locally on the site level. One important implication of this finding is that the onboarding outcomes cannot be always predicted and controlled by the central management.

We also learned that onboarding in globally distributed projects with legacy code is especially challenging, due to the difficulty to connect newcomers onboarded in offshore locations with the original developers, the difficulty to learn the legacy code, and potential onboarding strategy fragmentation due to the distribution. Our results also suggest that legacy projects, even co-located, require hands-on training and longer mentoring periods than new projects.

The use of the Bauer’s model facilitated in the analysis of the coverage and prevalence of the different onboarding functions. Among the case companies, the most common practice is coaching and mentoring, where the companies incorporate the new employees into existing teams and rely on the ability of the experienced team members to help the newcomers in their learning process. Some drawbacks of this approach is that this hinders productivity of the mentors, who experience frustration and stress associated with the inability to do their own work.

In our investigation, we also reflect on the repertoire of the onboarding strategies of the case companies, compared to the experience from companies from various disciplines collected by Bauer. While we are unable to judge on the sufficiency or efficiency of the documented strategies, we can see how the case companies can enhance their onboarding programs. For example, orientation was the most neglected function in our three cases.

Finally, we put forward the following future research directions:

- More empirical studies investigating onboarding in a holistic way. For future studies, we believe, it would be beneficial to conduct more empirical research, both investigating onboarding in a holistic way
(accounting for all onboarding function categories together and their interplay) and investigating separate onboarding function categories in depth.

- Quantitative studies that evaluate the effectiveness of onboarding strategies. By the time we conducted this investigation, we were not able to collect quantitative data. However, we believe that such data as retention of developers and performance of the onboarded developers and teams over time might allow understanding which onboarding strategies lead to better outcomes.

- Synthesis of the experiences with training as an onboarding function category. Training is the most studied onboarding function category in existing software engineering literature, with even venues fully dedicated to the topic (e.g., Conference on Software Engineering Education and Training, CSEE&T). However, to the best of our knowledge, there is no study that aggregates the existing empirical evidence. Therefore, we suggest to carry out a comprehensive secondary study to portray the state-of-the-art with respect to training as a part of onboarding strategies.

- Linking the findings from the motivation research and the onboarding research. Existing literature suggests that software developers are a distinct occupational group with particular drives or motivators [15]. Software developers are said to have a high need for growth and independence. It is fair to assume that software developers may, therefore, have also particular onboarding needs in comparison to other occupational groups. One important question for future research is then to understand the ways how to enrich the onboarding strategies to better outline the growth opportunities for the new software developers and provide them with the sense of independence as soon as possible.
EFFORT ESTIMATION IN LARGE-SCALE DISTRIBUTED AGILE SOFTWARE DEVELOPMENT: AN INDUSTRIAL CASE STUDY

1 INTRODUCTION

In today’s globalized market, software is increasingly developed in globally distributed projects [21, 49, 91, 172]. However, geographical, temporal, and cultural distances make coordination and communication more challenging in such projects, which may lead to more software defects [68] and schedule and budget overruns [90, 152].

The literature indicates that practitioners have fallen short of providing accurate and reliable effort estimates in both collocated and distributed projects. A secondary study by Moløkken and Jørgensen shows that 60–80% of the projects they surveyed had schedule and/or budget overruns [152]. More accurate effort estimates could help project managers to plan software projects in a better way, increasing the chances that projects do not get delayed or exceed budgets. More accurate effort estimates are not only associated with better effort estimation methods; to account for adequate predictors (i.e. size metrics and cost drivers) is equally important [144]. Identifying adequate predictors requires empirical research.

While a lot of research has covered effort estimation in software development projects in general [24, 111, 200], very little research has been conducted about effort estimation in large-scale distributed projects [57].

To address this gap, we conducted a longitudinal exploratory industrial case study, to investigate how effort estimation is performed in large-scale distributed agile projects and which factors (cost drivers) impact the accuracy of the effort estimates the most. Identifying these factors and their relationships with effort estimates would benefit managers at the case company, and other similar cases, to improve their estimation and planning processes. For the context of this work, we adopt Dikert et al.’s definition of large-scale projects, i.e. projects that involve at least 50 people or more than 10 teams [57]. The case is a large-scale distributed agile soft-
ware project with a large amount of complex legacy code at Ericsson\(^1\), a Swedish company that develops telecommunication-related products.

In this case study, we address the following research questions. The context for all research questions is large-scale distributed agile projects comprising a large amount of complex legacy code.

- **RQ1**: How is effort estimation carried out?
- **RQ2**: How accurate are the effort estimates?
- **RQ3**: Which factors impact the accuracy of the effort estimates?
- **RQ4**: How do the identified factors impact the accuracy of the effort estimates?

The main contribution of this chapter are described in the following:

- Understand and describe how effort estimation processes are managed in a large scale distributed agile project.
- Analysis of the effort estimates’ accuracy, and how re-estimation impacts estimates in a large scale distributed agile project.
- A better understanding of how certain factors affect effort estimates’ accuracy in a large-scale distributed agile project.

The remainder of the paper is organized as follows. Related work is presented and discussed in Section 2. Section 3 describes the research methodology employed in this case study. The results of the case study are presented in Section 4 and discussed in Section 5. Threats to validity are discussed in Section 6. Finally, Section 7 presents our conclusions and view on future work.

## 2 RELATED WORK

In this section, we discuss some primary and secondary studies about effort estimation in the context of global software development.

A review by Jørgensen [109] covering empirical research published 1990–2002, showed that expert estimation is the most commonly used estimation strategy in software projects. Jørgensen and Shepperd [111], furthermore noted that the proportion of studies on expert judgment based estimation is increasing, and that the available evidence does not support that the use of formal estimation models improves the estimation accuracy. More recent reviews [24, 200] on effort estimation in agile and global software development contexts also showed that the most frequently applied estimation methods are the ones that rely on subjective assessment by experts, such as planning poker, analogy, and use case points.

Britto et al.’s systematic literature review [24] identified a wide range of factors contributing to the effort of globally distributed projects, e.g.,

\(^1\) www.ericsson.com
socio-cultural, geographical and temporal distances. However, none of the primary studies provided detailed information about the effort estimation processes. A follow-up study by El Bajta et al. [66] corroborated these results. Usman et al. [200] identified team skills and prior experience as the most frequently used cost drivers in studies on effort estimation in agile projects.

A survey among practitioners by Britto et al. [25] indicated that the effort estimation processes in collocated and globally distributed agile projects are very similar, although the cost drivers depend on the development operational model (collocated or distributed). The main factors affecting effort estimation accuracy are requirements-related issues and communication overhead that is not properly accounted for.

In a similar survey, Usman et al. [201] found that similar effort estimation techniques are used in agile collocated and distributed contexts (mainly planning poker). There are differences in cost drivers though; geographic, socio-cultural and temporal distances were found to require more communication and coordination efforts in distributed projects than in collocated projects. In both contexts, requirements and project management-related issues impact negatively the accuracy of the effort estimates.

Lenarduzzi et al. [130] replicated a case study on functional size measures and effort estimation in a small agile project. Their results show that the effort estimated by developers was more accurate than the estimates obtained through functional size measures.

Evbotia et al. [69] investigated the challenges of scaling up the planning game in large-scale projects. In a qualitative case study at Ericsson they identified significant challenges in making long-term effort estimates in large-scale projects, because the associated product backlogs are too large. This made the teams involved in the study sceptical about effort estimation in general.

A case study by Tanveer et al. [195] showed that the accuracy of effort estimation in agile projects is affected by factors such as developers’ knowledge and experience, and the complexity of changes. The authors concluded that the explicit consideration of the aforementioned factors in the estimation process can support practitioners in obtaining more accurate estimates. Furthermore, they suggested that a tool combining expert knowledge and explicit consideration of cost drivers may improve the effectiveness of software effort estimation processes.

Based on the existing literature, we identified that:

- Expert judgment based effort estimation is the most commonly used approach in software industry.
- Similar approaches are used to estimate effort in both collocated and distributed contexts. The main differences are related to the cost
drivers; geographic and temporal distances demand more communication and coordination effort.

• There is a lack of in-depth research about effort estimation processes in large-scale distributed projects and the unique factors related to this context that may impact effort estimation accuracy.

This study fills the aforementioned gap through a longitudinal case study. It is unique in its combination of quantitative and qualitative data to analyze the effort estimation process from multiple perspectives, such as the impact of a multi-staged estimation process and various factors impacting effort estimation accuracy.

3 Research methodology

To address the research questions, we have conducted an exploratory longitudinal case study [181]. In this section, we describe the case, the unit of analysis and the data collection, preparation, and analysis processes.

3.1 The case and unit of analysis

The case studied in this chapter is a large-scale distributed agile project with an increasing degree of global distribution during the studied period. The actual project was selected through convenience sampling in consultation with company representatives.

The project is concerned with the development and maintenance of a large telecommunication software product at Ericsson. The product originated in Sweden and has evolved for about 20 years and comprises a considerable amount of legacy code. Many technical and methodological changes were introduced during this time, such as changing the programming language from C++ to Java and changing the software development methodology from plan-driven to agile. The product is part of a large business solution that comprises other products.

During the period covered in this study, the case involved 188 employees (15 product-level architects, 134 developers working in 24 teams and 15 other supporting roles) distributed across Sweden (eight software development teams), India (eleven software development teams), Italy (one software development team), USA (one software development team), Poland (two software development teams) and Turkey (one software development team).

The offshore locations were added in response to the growing demands for resources and to implement market-specific customizations. The last expansion was to Poland, where two teams were onboarded in early 2016. An expansion to India took place between late 2014 and late 2015, where
10 teams were onboarded in total. The site located in Turkey was decommissioned in 2014.

The developments teams are cross-functional and use agile practices in their daily work. Each team has from four to seven developers and a design lead, who is a senior developer. Due to the scale and level of distribution of the case, project managers use a mix of agile and plan-driven practices to manage and coordinate the work of the software development teams.

All software development teams receive an end-to-end responsibility for designing and implementing a task, which can be product customizations, trouble report fixing, product improvements, standardizations of market features and business use cases. These types of tasks resemble independent projects; they have specific start and end dates, responsible teams (one or more) and expected results. They can also be substantial in extent; the duration of a product customization, for example, varies from one to six months.

Software architects located in Sweden support all teams by responding to questions related to the product software architecture and by providing feedback on the teams’ work through code reviews. In urgent or particularly complex situations, the software architects also participate in actual coding.

We based our analysis on a subset of the collected data, focusing on product customizations (PCs), which are driven by Ericsson customers. Product customization projects are negotiated as fixed price contracts; failing to accurately estimate their effort may lead to budget overruns for the company. Accurate effort estimates for PC projects are therefore particularly important.

3.2 Data collection

To support methodological and data triangulation [181], we used the following data collection methods:

- **Archival research** – We analyzed managerial documents (plans, progress reports, time reports, solution specifications, effort estimation spreadsheets, process descriptions and team setup reports) of 60 PCs involving 18 different teams in total (9 in India, 7 in Sweden, 1 in Italy and 1 in the USA). We extracted the following data from these documents: effort estimation process description, effort estimates, actual effort, PC size, teams involved, team maturity, size of teams and customer type.
- **Unstructured individual interviews** - We held several unstructured interviews with a project manager to verify the consistence of the collected data.
• **Semi-structured individual interviews** – We interviewed two managers, a software architect, and a design lead to collect details about the effort estimation process employed in the case. We also talked to them to clarify the results of our data analysis, such as the main reasons that lead to the identified effort overruns. We covered with the interviews all roles involved in the case’s effort estimation processes. More details about the interviews are presented in Table 7.1.

Table 7.1: Description of semi-structured interviewee sample.

<table>
<thead>
<tr>
<th>Role</th>
<th>Approach</th>
<th>Duration</th>
<th>Experience</th>
</tr>
</thead>
<tbody>
<tr>
<td>System manager</td>
<td>face-to-face interview</td>
<td>70 minutes</td>
<td>9 years</td>
</tr>
<tr>
<td>Architect</td>
<td>face-to-face interview</td>
<td>45 minutes</td>
<td>7 years</td>
</tr>
<tr>
<td>Project manager</td>
<td>face-to-face interview</td>
<td>1 hour</td>
<td>15 years</td>
</tr>
<tr>
<td>Design lead</td>
<td>interview via Skype</td>
<td>45 minutes</td>
<td>10 years</td>
</tr>
</tbody>
</table>

Table 7.2 maps the research methods to the research questions. Note that we used both data collection methods for each of the research questions.

Table 7.2: Mapping RQs with the used data collection methods.

<table>
<thead>
<tr>
<th>RQ</th>
<th>Main method</th>
<th>Secondary method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Interviews</td>
<td>Archival research</td>
</tr>
<tr>
<td>2</td>
<td>Archival research</td>
<td>Interviews</td>
</tr>
<tr>
<td>3</td>
<td>Archival research</td>
<td>Interviews</td>
</tr>
<tr>
<td>4</td>
<td>Archival research</td>
<td>Interviews</td>
</tr>
</tbody>
</table>

3.3 **Data preparation**

We used a significant amount of data from different sources that was extracted using different data collection methods. To increase data reliability, we asked people involved with the data point (e.g., software architects or project managers) for clarification whenever we identified an issue or in-
consistency with a data observation and corrected the data observation, if necessary.

Furthermore, before running the quantitative part of the data analysis, it was necessary to aggregate the collected data. To do so, we created R scripts, which were used to create a unified data set.

3.4 Data analysis

To analyze the quantitative data (archival research), we calculated descriptive statistics, plotted charts to graphically identify trends in the data and employed inferential statistics (hypothesis testing and regression analysis) to answer RQ2–RQ4. To answer RQ1 and RQ3, we analyzed the qualitative data using light-weight qualitative analysis [167].

Before analyzing the accuracy of the effort estimates, we analyzed existing accuracy metrics. Traditionally, the Magnitude of Relative Error (MRE) is used frequently to calculate estimation accuracy. However, during the last decade, MRE has been criticized [116, 186], [110] for its uneven treatment of under- and over-estimation. BRE and BREbias are more balanced metrics, i.e. it evenly balances the over and underestimation. Due to this reason, Balanced Relative Error (BRE and BREbias) have been used in many recent studies on software effort estimation [137, 155]. BRE just calculates the magnitude of the estimation error, while BREbias measures both the size and the direction (over- or under-estimation) of the estimation error. Therefore, we used BRE (Equation 7.1) and BREbias (Equation 7.2) in our analysis of the effort estimates’ accuracy and bias.

\[
BRE = \frac{|actual\,\,effort - estimated\,\,effort|}{\min(actual\,\,effort,\,\,estimated\,\,effort)} \tag{7.1}
\]

\[
BREbias = \frac{(actual\,\,effort - estimated\,\,effort)}{\min(actual\,\,effort,\,\,estimated\,\,effort)} \tag{7.2}
\]

Before selecting an appropriate statistical test to analyze our results, we ran a Shapiro-Wilk normality test, which indicated that the collected data does not come from a normal population. Therefore, we applied non-parametric tests Wilcoxon signed-rank test [217] for RQ2 and Kruskal-Wallis [125] and Mann-Whitney [140] for RQ4). For the same reason, we used a non-parametric measure of Cliff’s delta [46] to measure effect size.

---

2 cran.r-project.org
4 RESULTS

4.1 RQ1: How is effort estimation carried out?

The aim of RQ1 is to explain how effort estimation processes are carried out in this very large distributed project. As shown in Figure 7.1, several people in different roles work collaboratively to perform effort estimation. First, we briefly describe these roles and the tasks that they perform in the effort estimation processes.

![Figure 7.1: Effort estimation process (PC = Product customization).](image)

- Customers: Ericsson customers demand new features in the product and have to approve the quoted price before development work on a product customization (PC) can begin.
- Customer Unit: Customer unit is part of Ericsson and is responsible for negotiations with the customers. It initiates the request for a new PC by specifying the requirements received from customers.
- System Managers: As stated previously, the product is part of a larger system comprising several products. System managers work at the overall system level. They receive request for new PCs from the customer unit, propose high level solutions and coordinate with the project managers and the architects of each concerned product in the system.
- Project Manager: Each product has a project manager, who is responsible for managing development teams across different sites and for planning, scheduling and coordinating the work on the PCs.
- Technical Writers: Technical writers are responsible for preparing PC documentation for the customers.
• Software Architecture Team (SAT): Each product has a team of architects, which is responsible for managing the evolution and integrity of the architecture. Besides proposing an initial solution and estimates for the requested PCs, they also provide design support, when required, to the development teams.

• Development Team: The development team is responsible for actually implementing and verifying the PCs.

The system managers, project managers, SAT and the development teams are part of the development unit. To fully understand the estimation processes in the case, we interviewed a representative from each of these roles. The interviewee sample (see Table 7.1) was selected by the unit manager, who was the contact person at Ericsson for this study.

Effort estimation is carried out at two levels in the case project, both using an expert judgment based estimation approach. First, a high level quotation estimate is prepared when a request for a new PC is initially analyzed. At this stage, it is not known which development team will actually develop the PC. Next, in a more detailed analysis phase, a more refined analysis estimate and a solution are proposed. At this stage, the development team that is going to develop the PC is known in most cases. We describe these estimation processes in the following.

**Quotation estimate (QE)** The process starts when a new PC request is initiated by the customer unit. The customer unit interacts with customers, and specifies the customer requirements. Once a new PC request is initiated, the following steps are performed:

• The system manager scans the new PC requests for an initial analysis. The PCs with relatively clear requirements are selected for further analysis, the remaining ones are sent back for further clarification.

• The selected PC requests are compared with previous PCs to identify any potential for reuse. The PCs are assigned to relevant system managers depending upon their availability. In case a PC request is identical to a previously developed PC, the customer unit is asked to order a resell instead of a new implementation.

• The assigned system manager analyzes the type of feature, and suggests a high level solution.

• The solution is shared with the SAT of the concerned product for further analysis. The SAT is also required to provide the QE. Normally, one member of the SAT leads this analysis.

• The QE is an interval estimate wherein the interval is marked by three points: min, max and average most likely estimate in hours. These intervals represent the size and complexity of PCs and are referred as small, medium, large and extremely large. Since the case has been going on for more than 15 years, an empirical classification
has been developed and evolved, which provides min, max and average hours for each interval estimate. The SAT lead uses this empirical classification to specify the relevant interval estimate for the assigned PC.

- The QE along with the high level solution are shared with the customer unit.
- The customer unit approves or disapproves the QE and the high level solution.

This process for preparing the quotation estimate takes up to four or five days. It is important to note that at the quotation stage the goal is to support the customer unit in deciding whether there is a strong business case in moving to the next phase of the PC development. The focus of effort estimation at this stage is on providing a best cost indication without spending too much time. Therefore, small estimation errors are not a problem at this stage.

**Analysis estimate (AE)** This process is carried out only for those PCs whose quotation estimates have been approved by the customer unit. The following steps are performed in this process.

- The system manager asks the SAT for the relevant products to prepare their respective designs, and to provide effort estimates in person hours.
- At this point, it is usually known which development team is going to develop the PC.
- One SAT member leads the work on each PC.
- The SAT lead specifies the design in detail. The SAT lead also estimates the effort, in person hours, required to develop and verify the PC. If the development team has a sufficient maturity and design competence (maturity level C and D, see Subsection 4.3 for details), the team’s design lead works together with the SAT lead during the design and estimation activities.
- The SAT lead presents the design and associated estimate to other SAT members for further discussion.
- The project manager reviews the estimate, and adjusts it, if required, based on the recent productivity and capacity of the concerned development team.
- The system manager forwards the design solution and corresponding estimates to the customer unit for approval.

4.2 **RQ2: How accurate are the effort estimates?**

The question aims to analyze the accuracy levels of the two estimates (i.e. QE and AE). This analysis is performed from the following two aspects:
• Overall trend analysis: To identify the dominant trends, we analyze how frequently the effort is over/under estimated corresponding to different estimation error levels (e.g., 25% to 50%). In this part, we do not use the mean or median of BRE and BREbias, as a single number (i.e. mean or median) does not fully convey the trend.

• Estimation error analysis: We analyze the accuracy of the effort estimates using BRE and BREbias, and compare the accuracy of the two estimates (QE and AE) to establish whether the re-estimation at the analysis stage (AE) improves the estimation accuracy over the original estimate (QE).

4.2.1 Overall trend analysis

As described in Subection 4.1, the QE is an interval estimate. Our results show that actual effort falls within the estimated interval in 26.7% of the cases (16 PCs), while 36.7% (22 PCs) are overestimated and 36.7% (22 PCs) are underestimated.

Tables 7.3, 7.4 and 7.5 summarize for both estimates, the frequency of over/under/accurately estimated PCs corresponding to different levels of estimation error expressed here as percentages. For QEs the PCs are almost evenly distributed in the three categories (overestimation, accurately and underestimated PCs), while in case of AEs underestimation is more common (50% PCs) as overestimation (16.7%). The gross overestimation instances are more common for QEs, where relatively less details about PCs are available, and the team that is going to develop the PC is not known.

Table 7.3: Accuracy distribution (overestimation) of the two estimates (QE and AE).

<table>
<thead>
<tr>
<th>Type</th>
<th>Above 100%</th>
<th>51–100%</th>
<th>26–50%</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>QE</td>
<td>10</td>
<td>3</td>
<td>5</td>
<td>18 (30%)</td>
</tr>
<tr>
<td>AE</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>10 (16.7%)</td>
</tr>
</tbody>
</table>

4.2.2 Estimation error analysis

In this part we analyze and compare the accuracy of the two effort estimates using BRE and BREbias measures. The results are displayed in Table 7.6. The QEs’ magnitude of error (mean BRE 1.26, median BRE 0.69) is considerably higher than the AEs’ magnitude of error. The mean and median

3 for QE we used mid value, which is the most likely estimate, to compute BRE.
Table 7.4: Accuracy distribution (accurately) of the two estimates (QE and AE).

<table>
<thead>
<tr>
<th>Type</th>
<th>No. of accurately estimated PCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>QE</td>
<td>20 (33.3%)</td>
</tr>
<tr>
<td>AE</td>
<td>20 (33.3%)</td>
</tr>
</tbody>
</table>

Table 7.5: Accuracy distribution (underestimation) of the two estimates (QE and AE).

<table>
<thead>
<tr>
<th>Type</th>
<th>26–50%</th>
<th>51–100%</th>
<th>Above 100%</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>QE</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td>22 (36.7%)</td>
</tr>
<tr>
<td>AE</td>
<td>6</td>
<td>11</td>
<td>13</td>
<td>30 (50%)</td>
</tr>
</tbody>
</table>

BREbias values for both estimates show that the estimates are biased towards underestimation, i.e. optimism. The underestimation bias, however, is more significant for AEs (mean BREbias 0.41, median BREbias 0.26).

Table 7.7 displays the results of a Wilcoxon signed-rank test to see if the differences between the two estimates, for our sample of 60 PCs, are statistically significant. The statistical tests are conducted in SPSS, and we report all relevant data from SPSS, and also the effect size measure (Cliff’s delta), when applicable. The results show a statistically significant difference (p = 0.04) between the BREbias of the two estimates with small to medium size effect (Cliff’s delta -0.14). However, the p-value of the test for the BRE of the two estimates is slightly over 0.05 (0.06) and therefore not statistically significant. The results in Table 7.6 and 7.7 show that the AEs are more accurate (median BRE 0.49) than the QEs, but contain significantly higher optimism (i.e., underestimation) bias.

Table 7.6: Mean and Median BRE and BREbias for both estimates (QE and AE).

<table>
<thead>
<tr>
<th></th>
<th>BRE QE</th>
<th>BRE AE</th>
<th>BREbias QE</th>
<th>BREbias AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>1.26</td>
<td>0.81</td>
<td>0.25</td>
<td>0.41</td>
</tr>
<tr>
<td>Median</td>
<td>0.69</td>
<td>0.49</td>
<td>0.05</td>
<td>0.26</td>
</tr>
<tr>
<td>Std. Deviation</td>
<td>1.48</td>
<td>0.97</td>
<td>1.93</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Table 7.7: Testing the two estimates (QE and AE) for significant differences.

<table>
<thead>
<tr>
<th>Source</th>
<th>N</th>
<th>Mean rank</th>
<th>Sum of ranks</th>
<th>Z</th>
<th>P-value</th>
<th>Size effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRE AE – BRE QE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative Ranks</td>
<td>34</td>
<td>33.32</td>
<td>1133</td>
<td>-1.87</td>
<td>.06</td>
<td>-0.15</td>
</tr>
<tr>
<td>Positive Ranks</td>
<td>25</td>
<td>25.48</td>
<td>637</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ties</td>
<td></td>
<td>1</td>
<td></td>
<td>-2.03</td>
<td>.04</td>
<td>0.14</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BREbias AE – BREbias QE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative Ranks</td>
<td>20</td>
<td>30.80</td>
<td>616</td>
<td>-2.03</td>
<td>.04</td>
<td>0.14</td>
</tr>
<tr>
<td>Positive Ranks</td>
<td>39</td>
<td>29.59</td>
<td>1154</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ties</td>
<td></td>
<td>1</td>
<td></td>
<td>-1.87</td>
<td>.06</td>
<td>-0.15</td>
</tr>
<tr>
<td>Total</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3 RQ3: Which factors impact the accuracy of the effort estimates?

Through the interviews, we identified the following factors that potentially impact the accuracy of the two effort estimates:

- **PC size** – Understandably, the estimation of larger PCs is perceived to be more challenging. The interviewees observed that PC size, measured in actual effort (work hours), beyond a certain level brings with it all the complexities of scale, and therefore impact the accuracy of the effort estimates. Using the actual effort spent, the PCs were classified based on the classification mechanism described in Section 4.1 into four types: small, medium, large and very large. In our regression analysis, the variable PC size is used as an ordinal variable with values 0 (small), 1 (medium), 2 (large) and 3 (very large).

- **Customer priority** – A set of PCs (26) in our sample belong to one large customer project. The case company ensures a dedicated team capacity for the implementation of these PCs, in return for the guarantee that the company will be paid for the work done on these PCs. The provision of dedicated capacity for these PCs is meant to prioritize them during planning phase. They are not required to wait for the availability of capacity to start the work. Therefore, the work is
initiated relatively quickly on most of the high priority PCs after the quotation approval by the customer unit. These 26 PCs are categorized as PCs having a customer with high priority, while the remaining 34 as PCs from customers with a normal priority.

- **Maturity of the development team** – The development teams, working across the globe, have varying levels of maturity. As teams become more mature, they work more independently and need less support from the product level architects. We use the same maturity levels as Britto et al. [27], who studied the same case company.
  - **Maturity level A** – Teams at level A have very little understanding about the product’s code and architecture. Therefore, they need much support and guidance from the architects, even for less complex PC projects. Typically, newly onboarded teams are at level A.
  - **Maturity level B** – Teams at level B can implement non-complex tasks independently without much help from the product level architects. The architects still review most of the design and code.
  - **Maturity level C** – Teams at level C have a good architectural understanding and are able to implement complex solutions. These teams perform code reviews and approvals independently. When critical components are affected, the product level architects are responsible for approval. The architects also support these teams with the design of the technical solution.
  - **Maturity level D** – Teams at level D are very experienced, and are able to autonomously develop complex solutions that affect or include critical components. Their code does not need the approval from the product level architects.

In our sample of 60 PCs, 19 PCs were developed by level A teams, 12 by level B teams, 26 by level C teams, and 3 by level D teams. The variable team maturity is used as an ordinal variable in our regression analysis with the four values: 0 (maturity level A), 1 (maturity level B), 2 (maturity level C), 3 (maturity level D).

- **Multi-site development** – In our sample, 6 PCs were developed in a multi-site arrangement wherein development teams in different geographical locations collaboratively worked together to develop the PCs. Multi-site development is also perceived as a factor impacting the actual effort spent, and thus the accuracy of the effort estimates. However, we decided not to include this factor in our regression analysis for two reasons: 1) Very few PCs were in this category, and 2) the multi-site PCs were particularly large. It would therefore not be possible to differentiate the impact of PC size and multi-site setting. We,
however, analyze the impact of multi-site development in more detail in Section 4.4.4.

We performed multiple linear regression to investigate whether the factors identified above significantly impact the accuracy of the effort estimates. We used simple linear models to understand the relationship between the effort estimates’ accuracy and the factors described above. We applied stepwise regression in SPSS using backward elimination with an alpha value of 0.1 to remove variables.

4.3.1 Regression model of quotation estimates (QEs)

The results of the regression analysis showed that PC size ($p < 0.005$) and team maturity ($p = 0.007$) have a statistically significant relationship with BREbias of QEs (see Table B.1 and B.2 in Appendix B for details), while PC priority ($p = 0.863$) has no such relationship.

Given the details in Table B.2, the regression model is described as:

\[
\text{BREbias of QE} = -0.73 + 1.22 \times \text{PC Size} - 0.56 \times \text{Team Maturity} \quad (7.3)
\]

Model (7.3) suggests that the QEs are overestimated by 73% for the small size PCs (PC size = small = 0) and immature teams (Team Maturity = Level A = 0). However, increases in PC size increase the BREbias of QEs, i.e. underestimation bias, while increases in team maturity decrease the BREbias of QEs, i.e. overestimation bias. In other words mature teams are more likely to complete the work within the estimated time. These factors are investigated in detail in Section 4.4 and are further discussed in Section 5.

Model (7.3) is the one with highest value (0.40) of adjusted $R^2$ (see Table B.3 in Appendix B). The value of adjusted $R^2$ does not necessarily increase when more predictors are added in the model, and thus is a suitable measure in identifying appropriate predictors for a model. Including variable PC Priority ($p = 0.863$) slightly reduces the value of adjusted $R^2$.

4.3.2 Regression model of analysis estimates (AE)

Our regression analysis showed that PC size ($p = 0.001$) and priority ($p = 0.028$) have a statistically significant relationship with BREbias of AEs, while team maturity ($p = 0.929$) has no such relationship. The regression details are shown in Tables C.1 and C.2 in Appendix C.

Given the details in Table C.2, the regression model is described as:

\[
\text{BREbias of AE} = -0.07 + 0.57 \times \text{PC Size} - 0.69 \times \text{Customer Priority} \quad (7.4)
\]
Model (7.4) shows that for small sized (PC size = small = 0) PCs with normal priority (customer priority = normal = 0), effort is slightly overestimated by 7%. An increase in PC size (PC size = medium = 1 or PC size = large = 2 or PC size = very large = 3) results in an increase in underestimation bias. However, the negative sign of priority in model (7.4) indicates that an increase in priority from normal to high (i.e. customer priority = normal = 0 to customer priority = high = 1) decreases the underestimation bias or leads to an overestimation bias. In other words, the high priority PCs incur lesser magnitudes of effort overruns with respect to the analysis stage estimates. However, PC size is still a huge factor even with high priority PCs. When PC size becomes large or very large, the overall result is still a considerable underestimation bias. These factors are further analyzed in detail in Section 4.4, and further discussed in Section 5. Team maturity is not significantly impacting BREbias of AEs, since the estimators at the analysis stage are normally aware of which teams are going to develop a PC. Therefore, Team maturity already is accounted for in the analysis stage estimates.

Model (7.4) is the one with highest value (0.40) of adjusted $R^2$ (see Table C.3 in Appendix C). Including variable Team Maturity ($p = 0.929$) reduces the value of adjusted $R^2$.

4.3.3 Regression assumptions

The examination of the residuals of both models revealed that the regression assumptions are not violated, with one minor exception. The histograms and normal plots show some minor deviations from the normal distribution (see Figures B.1–B.3 and C.1–C.3 in Appendices B and C). Besides visual analysis using these plots, we also tested these assumptions using relevant tests.

- Independence of residuals was tested using a Durbin-Watson test [59]. The results for both models were in the required range of 1.5 and 2.5 (2.01 for model (7.3) and 2.09 for model (7.4)).
- Constant variance in residuals (Homoscedasticity) was tested using a Breusch-Pagan and Koenker tests [23]. For both models, the null hypothesis of homoscedasticity was not rejected due to p-values > 0.05: Breusch-Pagan (model (7.3), $p = 0.08$; model (7.4), $p = 0.41$), Koenker (model (7.3), $p = 0.28$; model (7.4), $p = 0.75$).
- Normal distribution of residuals was tested using a Kolmogorov-Smirnov test [121]. For both models the tests showed that the residuals are normally distributed, i.e. the null hypothesis of normality could not be rejected (model (7.3), $p = 0.20$; model (7.4), $p = 0.08$). According to a Shapiro-Wilk test, the residuals of model (7.3) are normally distributed ($p = 0.17$), but not the residuals of model (7.4) ($p = 0.003$).
Furthermore, tolerance values (see Table B.2 and C.2 in Appendix B and C respectively) show the absence of multicollinearity for both models. As for the outliers diagnostics, only one observation was outside the three sigma limit in case of model (7.3). The maximum cook’s distance [51] for any observation was 0.11 (much less than critical value of 1), indicating that there are no highly influential observations. Likewise for model (7.4), there is only one observation that is outside the three sigma limit, and the maximum cook’s distance was 0.31.

Besides BREbias, we also attempted to apply regression analysis using BREs of both QEs and AEs. In case of BRE of QEs, regression assumptions of homoscedasticity and normality were violated. For AEs, the analysis did not find any significant relationship between the outcome and independent variables.

4.4 RQ4: How do the identified factors impact the accuracy of the effort estimates?

This question aims to investigate in detail how the factors, identified as statistically significant above, impact the accuracy of the effort estimates.

4.4.1 PC size

The 60 PCs included in this study are of varying sizes in terms of actual effort spent. These PCs are divided into four categories (see Table 7.8) using the intervals of QEs (see Section 4.1). The aim of this question is to investigate in depth how PC size impacts the accuracy of the two effort estimates.

Table 7.8: Mean and median actual effort for all 60 PCs in work hours grouped by PC size.

<table>
<thead>
<tr>
<th>PC size</th>
<th>No.</th>
<th>Mean (hours)</th>
<th>Median (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>10</td>
<td>284.9</td>
<td>290</td>
</tr>
<tr>
<td>Medium</td>
<td>28</td>
<td>833.1</td>
<td>699</td>
</tr>
<tr>
<td>Large</td>
<td>12</td>
<td>1931.8</td>
<td>1561</td>
</tr>
<tr>
<td>Very Large</td>
<td>10</td>
<td>5735.5</td>
<td>4811.5</td>
</tr>
</tbody>
</table>

Quotation estimate (QE) Figure 7.2 depicts the QEs’ BREBias (vertical axis) for all 60 PCs that are sorted from left to right in increasing order of actual effort spent. The bars below 0 represent overestimated PCs (i.e.
estimate > actual), while above 0 are underestimated PCs (i.e. estimate < actual).

The results clearly show a contrasting pattern with respect to over- and underestimation; smaller PCs tend to be overestimated, while larger PCs tend to be underestimated.

Figure 7.2: BREbias of QEs of PCs arranged on actual effort.

Figure 7.3: BREBias boxplots by size for QE.
Grouping BREBias by PC size shows that an increase in size seems to be related to an increase in estimation error and underestimation bias (see Figure 7.3). We therefore applied a Kruskal-Wallis test \[ \text{[125]} \] to see whether there are statistically significant differences between the QE’s BREbias of the four PC sizes. The results shows that the BREbias of QEs correlates significantly with PC size \((p < 0.0001 \text{ at } \alpha = 0.05)\).

![Figure 7.4: BREbias of AEs of PCs arranged on actual effort.](image)

**ANALYSIS ESTIMATE (AE)** AEs show a similar trend, though the division of over- and underestimated PCs is not that clear (see Figure 7.4). Some smaller PCs are underestimated, and a few larger PC are slightly overestimated. However, underestimation becomes more frequent as we move to large and very large PCs.

Figure 7.5 shows AEs’ BREBias grouped by PC size. The results show that underestimation bias increases with the increase in PC size. A Kruskal-Wallis test showed that there are statistically significant differences between the BREbias of different PC sizes \((p < 0.0022 \text{ at } \alpha = 0.05)\).

For both estimates (QE and AE), there are statistically significant differences in accuracy levels for different PC sizes. The differences are pronounced though for QEs.

Table 7.9 lists the mean and median BREbias of the two estimates for the four PC sizes. The results show two trends: 1) For both QEs and AEs, an increase in size is related to an increase in underestimation bias. The increase in bias is larger for QEs, indicating that PC size impacts QEs more strongly; 2) For almost all PC sizes, the magnitude of estimation error
Figure 7.5: BREBias boxplots by size for AE.

decreases as we move from QEs to AEs, i.e. the re-estimation at the analysis stage (AE) improves the accuracy of the effort estimates.

Table 7.9: Mean and Median BREBias for both estimates (QE and AE) grouped by PC size.

<table>
<thead>
<tr>
<th>PC size</th>
<th>Mean (QE)</th>
<th>Mean (AE)</th>
<th>Median (QE)</th>
<th>Median (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>-1.18</td>
<td>-0.40</td>
<td>-0.99</td>
<td>-0.23</td>
</tr>
<tr>
<td>Medium</td>
<td>-0.34</td>
<td>0.35</td>
<td>-0.21</td>
<td>0.34</td>
</tr>
<tr>
<td>Large</td>
<td>1.31</td>
<td>0.76</td>
<td>0.40</td>
<td>0.29</td>
</tr>
<tr>
<td>Very Large</td>
<td>2.10</td>
<td>0.97</td>
<td>1.30</td>
<td>0.56</td>
</tr>
</tbody>
</table>

4.4.2 Customer priority

Our dataset comprises 26 PCs with high priority (1 small, 10 medium, 7 large, 8 very large) and 34 PCs with normal priority (9 small, 15 medium, 7 large, 3 very large). To make the samples and the results comparable, we excluded the small PCs, since there was only 1 small PC with high priority. The mean and median BREBias of the remaining 50 PCs (25 PCs
each with high and normal priority) are summarized in Table 7.10. These results show the following interesting patterns:

- The estimation accuracy improves considerably as we move from quotations (QE) to analysis (AE) stage estimation for high priority PCs. However, for normal priority PCs it is other way around.
- At the quotation stage, the estimates (QE) of normal priority PCs are more accurate as compared to the high priority PCs.
- At the analysis stage however, the estimates (AE) of the high priority PCs are more accurate.

These results indicate that there is a relation between customer priority and estimation accuracy. However, a Mann-Whitney test [140] did not detect a statistically significant difference in the distributions of BREbias of QEs and AEs across the two priorities (p < 0.46 for QEs and p < 0.23 for AEs; α = 0.05).

Table 7.10: Mean and median BREBias for both estimates (QE and AE) grouped by PC priority.

<table>
<thead>
<tr>
<th>Customer priority</th>
<th>Mean (QE)</th>
<th>Mean (AE)</th>
<th>Median (QE)</th>
<th>Median (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>0.68</td>
<td>0.31</td>
<td>0.32</td>
<td>0.20</td>
</tr>
<tr>
<td>Normal</td>
<td>0.41</td>
<td>0.84</td>
<td>0.10</td>
<td>0.54</td>
</tr>
</tbody>
</table>

4.4.3 Team maturity

Team maturity varies from level A (lowest level) to level D (highest level), see Section 4.3. To facilitate an analysis, we categorized teams at levels A and B as immature teams and teams at levels C and D as mature. PCs of all sizes are developed by these two types of teams.

- Number of PCs by immature teams: small (6), medium (14), large (7), very large (4)
- Number of PCs by mature teams: small (4), medium (14), large (5), very large (6)

Table 7.11 shows that the mean effort estimate accuracies are better for mature teams, but quite similar for the medians. As for the estimation bias, the results show a considerable reduction in the underestimation bias for mature teams. Together, it means that immature teams are more likely to incur larger effort overruns as compared to the mature teams. However, a Mann-Whitney test did not show any statistically significant differences in the distributions of BREbias for QEs and AEs across two types of teams.
Table 7.11: Mean and median BREBias for both estimates (QE and AE) for immature and mature teams.

<table>
<thead>
<tr>
<th>Team Maturity</th>
<th>No. of PCs</th>
<th>Mean (QE)</th>
<th>Mean (AE)</th>
<th>Median (QE)</th>
<th>Median (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immature teams</td>
<td>31</td>
<td>0.68</td>
<td>0.60</td>
<td>0.03</td>
<td>0.33</td>
</tr>
<tr>
<td>Mature teams</td>
<td>29</td>
<td>-0.20</td>
<td>0.26</td>
<td>0.08</td>
<td>0.25</td>
</tr>
</tbody>
</table>

4.4.4 Multi-site development

Six PCs in our dataset were developed by geographically distributed teams (referred here as multi-site development), while the remaining 54 were developed by collocated teams with remote support provided by software architects located in Sweden. The development of these 6 PCs is led by a team at one site. The teams in other sites contribute by developing relatively smaller and specific parts due to special competencies at these sites. Multi-site development could also have an impact on the accuracy of the effort estimates.

The six multi-site development PCs have mainly large sizes (very large: 3, large: 2, medium: 1), whereas the PCs developed in a collocated setting have all types of sizes. To make a fair comparison, we excluded all small PCs, leaving behind 44 PCs of medium, large and very large size for the collocated case. Table 7.12 summarizes effort estimation accuracies for collocated and multi-site PCs and shows that both mean and median BREbias are higher for multi-site PCs.

These results indicate that multi-site development in the case leads to relatively larger effort overruns. However, a Mann-Whitney test did not find the differences in the distribution of the BREbias for both QEs and AEs (44 collocated PCs and 6 multi-site PCs) to be statistically significant across two types of PCs, i.e. collocated and multi-site. This may be due to the very small number of the PCs in the multi-site category.

Moreover, in order to see whether multi-site development phenomena has a statistically significant impact on the results of the statistical tests reported in 4.2 (Table 7.7), Section 4.4.1, 4.4.2 and 4.4.3, we removed the multi-site PCs from our sample, and re-performed these tests. All tests returned similar results with minor changes in the resulting p-values. The changes were so insignificant that they did not lead to any reversal in the corresponding null hypotheses testing results.
Table 7.12: Mean and median BREBias for both estimates (QE and AE) for collocated and multi-site PCs.

<table>
<thead>
<tr>
<th>PC type</th>
<th>Mean (QE)</th>
<th>Mean (AE)</th>
<th>Median (QE)</th>
<th>Median (AE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Co-located</td>
<td>0.41</td>
<td>0.41</td>
<td>0.09</td>
<td>0.34</td>
</tr>
<tr>
<td>Multi-site</td>
<td>1.51</td>
<td>1.79</td>
<td>1.28</td>
<td>1.18</td>
</tr>
</tbody>
</table>

4.5 Results from the interviews

Besides understanding the effort estimation processes used in the case, the interviews were also used to discuss the results with the participants. Information about the four interviewees are provided in Table 7.1 in Section 3. The interviewees were asked about the following: 1) their background and experience in the company and the case, 2) their role in the effort estimation processes, 3) how various factors impact the accuracy of the effort estimates, and 4) the main reasons for effort overruns. The information will be used in the discussion section to further explain the results presented above.

5 Discussion

In this section we further discuss the results presented in Section 4 in the light of our discussion with interviewees.

5.1 Effort estimation processes (RQ1)

Our interviews show that it is mainly the scale of a PC project that makes it challenging to estimate and plan. Scale includes a number of factors beyond PC size, such as the number of sites involved, the number of stakeholders involved in the process, the size and complexity of the legacy code etc. The coordination between different types of stakeholders involved in the estimation processes also introduces challenges (see Section 4.1 for stakeholders details). Research shows that coordination challenges exacerbate issues in multi-team projects [57]. Despite these challenges, a reasonable proportion of PCs (1/3, see Table 7.4) are accurately estimated within a 25% error margin at both quotation and analysis stages.

The involvement of teams in the analysis stage estimation process is an important practice. The interviewed architect and project manager both suggested that it helps in arriving at better estimates. Whenever possible, the teams should provide their input in the estimation process. However, such a practice is only feasible with mature teams, i.e. teams including
team members with sufficient expertise and knowledge about the product’s design and architecture. The newly onboarded teams need more time to come to this level.

5.2 Accuracy of estimates (RQ2)

Our analysis showed that, overall, underestimation is the dominant trend at both quotation and analysis stages. This is in line with the results in other estimation studies [152]. The interviewees attributed the inaccuracies in the effort estimates mainly to the following challenges that they encounter in this very large-scale distributed agile project:

- Requirements related issues such as lack of details and changes in the requirements.
- Lack of expertise of newly onboarded teams results in delays.
- Dependencies (such as for code reviews) on specific human resources (e.g., product architects) introduce delays.
- Project scale and distribution across multiple sites.
- Underestimating the technical complexity of some large PCs.

All interviewees stressed the importance of better requirements to be able to estimate more effectively. There is lot of uncertainty at the quotation stage, and late changes/additions in the requirements eventually results in underestimation.

Since the goal of the quotation stage estimation (QE) is to provide a relatively quick estimate to the customer unit, expectations on accuracy are not as high as they are for analysis stage estimates (AE). Our results show that the AEs actually are more accurate than the QEs. However, underestimation bias increases at the analysis stage. The interviewees suggest that the availability of more detailed information and knowledge of the teams and their input at the analysis stage might have increased the estimators’ optimism, resulting in relatively higher underestimation. Underestimation bias is relatively less evident at quotation stage, where in 30% of the PCs the effort is overestimated (see Table 7.3). In these cases, the estimators are more conservative in their estimates mainly due to a high level of uncertainty, lack of detail in the requirements, and that the PCs have not yet been assigned to a development team (i.e. the development team’s maturity level is unknown).

5.3 Factors affecting the effort estimates (RQ3 and RQ4)

We identified four factors that impact the accuracy of the effort estimates (Section 4.3) and analyzed how these factors affect the effort estimates (Section 4.4). We now discuss these factors in the light of the interview results.
5.3.1 **PC size**

It is more challenging to estimate larger PCs. We found statistically significant differences in the distribution of BREbias, for both types of effort estimates (QE and AE), across PCs of different sizes. For smaller PCs there is a tendency to overestimate, while for larger and very large PCs our results show a large underestimation bias. The interviewees attribute this to the inherent difficulty in estimating large PCs, whereby estimators underestimate the likely complexities and technical challenges. Moreover, the development of large PCs involve more unanticipated challenges and risks related to project management, coordination, design and architecture, resulting in potentially large delays. Moløkken-Østvold and Furulund [153] also identified a tendency of larger effort overruns for projects with larger actual effort.

5.3.2 **Team maturity**

Team maturity also impacted the effort estimates. Our results show that PCs developed by mature teams have more accurate effort estimates. In the present case 11 teams were onboarded relatively recently. They are the most immature teams in our dataset, and are incurring higher effort overruns than the more mature teams. In order to mature, newly onboarded teams require a lot of time and mentoring support from product architects in Sweden [27]. The mentoring support provided by the product architects is critical to achieve technical consistency, which otherwise is hard to maintain in such projects that scale up involving several agile teams at different levels of maturity [57].

At the quotation stage, the estimators do not know which team(s) will work on the PC being estimated. This might lead them to estimate pessimistically. The interviewees suggested that mature teams perform better than the estimates due to a better understanding of the product architecture and the high technical competence. At the analysis stage estimation, team maturity is not a significant factor according to our regression analysis (see Section 4.3). This might be due to the fact that the team’s maturity is already accounted for in the AEs, as the estimators are aware of the team which is going to work on the PC being estimated.

5.3.3 **Customer priority**

Customer priority was also identified as a factor impacting the effort estimates’ accuracy. The results show a contrasting pattern, i.e. effort estimates of PCs with high priority are more accurate at analysis stage (AE) and less accurate at quotation stage (QE), while it is the other way around in case of PCs with normal priority. The interviews revealed that for high priority PCs there is relatively more emphasis to shorten lead times by ensuring
dedicated capacities for these high priority PCs. Moreover, as the work is initiated relatively quickly on most of the high priority PCs, more is known when the estimation at the analysis stage is performed. More information implies less uncertainty, which could be a reason for the relatively better analysis stage estimates (AE). The interviewees attributed the relatively higher inaccuracy of QEs for high priority PCs mainly to the late changes in requirements after the quotation stage, which is much more common in case of high priority PCs.

5.3.4 Multi-site development

Six PCs in our dataset were developed in multi-site settings. It is too small a number to perform a generalizable quantitative analysis. Nevertheless, in our limited sample the multi-site PCs have a stronger underestimation bias than collocated PCs, i.e. they incur larger effort overruns (as compared to the estimates). The interviewees attribute this to the inherent difficulties related to communication and coordination present in multi-site arrangements. Global barriers related to time, distance and culture in multi-site development have been identified as important cost contributors in other studies as well [25]. Herbsleb and Mockus [90] found that multi-site development introduces significant delays during the task development. Dikert et al. [57] identified that managers in several organizations find it challenging to create and estimate user stories in large scale distributed agile projects. This impact of multi-site development on effort estimation needs to be further investigated, preferably with a larger dataset.

6 Threats to validity

For our discussion, we use the classification of threats to validity by Runeson and Höst [181]. Furthermore, we also discuss conclusion validity [198].

Reliability is related to the repeatability of a study and how data is collected and analyzed [181]. To minimize this threat, we designed and followed an explicit, detailed case study protocol, following the guidelines by Runeson and Höst [181]. Furthermore, several researchers were involved in the design and execution of our investigation to minimize dependencies based on particular individuals. In addition, our observations and findings were verified with the company representatives to avoid false interpretations.

Internal validity is related to factors that researchers are unaware of or cannot control regarding their effect on the variables under investigation [181]. The effort estimates’ accuracy, how the estimates are obtained (estimation process) and actual effort in projects are constructs influenced by different factors, such as team maturity. It is therefore difficult to identify causal relationships between them, since there are many confounding fac-
tors. To mitigate this threat, we accounted for factors that are reported by related literature and also by interviewing people in the case company who are involved with effort estimation in the studied case (data triangulation). Regarding the qualitative part of our research (semi-structured interviews), the main internal validity threats are investigator bias and interviewee bias. To mitigate these threats, three researchers were involved with the design of the interview guides (investigator triangulation). We mitigated interviewee bias by interviewing people with different roles (data triangulation).

**Construct validity** reflects how well the measures used actually represent the constructs the study intends to measure [181]. The main threat of this category related to our investigation is the mono-method bias (only one method to measure a construct). Although just one measurement method was used to measure each construct, we cross checked different documents to increase the reliability of the collected data, with special focus on the collected actual effort. For example, we compared the data on progress reports and time reports to see whether the hours reported were in line with the number of people working in a given week in a particular task. Finally, we also conducted unstructured interviews with a project manager to further verify the consistency of the collected data.

**External validity** is concerned with the generalizability of the findings [181]. Since we employed the case study method, our findings are strongly bounded by the context of our study. In addition, the investigated case involved only one project in one company. To mitigate this threat, we made an attempt to detail the context of our study as much as possible, complying with corporate confidentiality concerns. To mitigate this threat, we made an attempt to describe the context of our study in as much detail as possible. This makes it easier to relate the present case to similar cases.

**Conclusion validity** is concerned with the correctness of conclusions regarding relationships in the analyzed data [198]. The main threats of this category related to our investigation are the low reliability of measures due to noise and low statistical power. To mitigate the first threat, we conducted an unstructured interview with a project manager to verify the consistency of the collected data. Regarding statistical power, we investigated the largest number of tasks possible. However, it is important to emphasize that it is just a sample and does not cover all product customization tasks carried in the investigated case.

### 7 CONCLUSION

This chapter reports the results of a case study conducted in Ericsson with the aim to investigate the effort estimation in a large-scale distributed agile project. Ericsson is a large multi-national telecommunication vendor with headquarters in Sweden. The case is related to a large telecommunication
product, which is being evolved for over 15 years by agile teams distributed across Sweden, India, Italy, USA, Poland and Turkey. The studied product is part of a large business solution that includes other products as well. We used archival research to collect data about 60 Product Customization (PC) tasks involving 18 different teams in Sweden, India, Italy and USA. The collected data included effort estimates, actual effort, teams involved, team maturity and customer type. We conducted interviews to understand the context, and also to discuss and explain the results.

**rq1: effort estimation process** Effort estimation process is carried out at two stages in the case: quotation and analysis. At the quotation stage, the system manager along with the SAT prepares the QE to support the customer unit in deciding whether there is a business case in approving the PC for development. At the analysis stage, a more detailed analysis is performed of those PCs whose QEs have been approved by the customer unit. The assigned SAT member leads this analysis to propose a solution and effort estimate of the PC. At this stage, in most case the development team is already assigned to the PC under analysis. If the assigned team has design competence, the SAT member involves the team’s design lead in the analysis process.

**rq2: accuracy of the effort estimates** The results showed that underestimation is the more frequent trend at both quotation and analysis stages. The estimates at the analysis stage (AEs) were found to be more accurate than the QEs. However, they have significantly higher optimism bias as compared to QEs.

**rq3 and rq4: factors impacting effort estimates** Based on our analysis of 60 PCs from one project within one organization, we identified PC size, team maturity, PC priority and multi-site development as factors that impact the accuracy of the effort estimates. Through regression analysis we found that: 1) PC size and team maturity have statistically significant relationship with BREbias of QEs, and 2) PC size and PC priority have a statistically significant relationship with BREbias of AEs. The results indicate that PC size has a strong influence on the effort estimates’ accuracy and bias, i.e. increase in PC sizes was found to be strongly related to the rise in the underestimation bias. We also identified that the mature teams, due to their experience and technical competence, are less likely to incur large delays. Furthermore, PCs with high priority in terms of dedicated capacity have better analysis stage estimates mainly due to the early start of work on them. Furthermore, the results also indicated that the multi-site PCs have larger magnitude of effort overruns as compared to the co-located PCs. The identified correlations between effort estimates and these factors
(e.g. PC size, team maturity, customer priority and multi-site development) needs to be further investigated before drawing any conclusions.

These results have important implications for practice. Software practitioners, working in similar contexts, should consider the following aspects when estimating:

- As underestimation is the more dominant trend, it is important to consider factors (e.g., PC size, team maturity, etc.) that could potentially add delays in the project. The identification and consideration of these factors would be helpful in reducing over-optimism bias.
- It is important to involve all concerned stakeholders (e.g., system managers, project manager, architects, design lead) in the estimation process. Specifically, software architects have a pivotal role in the estimation process due to their understanding of the product’s architecture, which allows them to effectively perform an impact analysis of new requirements/tasks at the system level.
- Re-estimation at the analysis stage improves the effort estimates. At this stage estimators know more about the requirements and the assigned team(s). These refined estimates at the analysis stage support project managers in monitoring the progress on the task development.
- PC size was found to be strongly related to the effort estimates’ accuracy and bias. Large sized requirements/tasks are more likely to incur large effort overruns.
- Mature teams should be involved in the effort estimation process as they have architectural knowledge and expertise.
- Immature teams need more time to complete the tasks, and hence are more likely to incur effort overruns.
- Multi-site development, wherein geographically distributed teams collaboratively work on a task, exacerbates the underestimation bias.

As for research, the identified factors need to be further investigated in other contexts. The practitioners in the case use expert judgment to estimate the development effort. Expert judgment is the most widely used estimation technique [109]. We plan to investigate how the explicit consideration of relevant factors compliments expert judgment in arriving at better and informed effort estimates.
1. Please provide your background.
2. How does the recruiting process work?
   a. What are the characteristics that are mandatory for a candidate to be hired?
   b. What are the characteristics that are not appreciated in new candidates?
   c. Is the recruiting process evaluated?
3. How does the learning process work?
   a. What are the scenarios wherein developers need to be trained?
   b. What are the main learning needs (technical knowledge, methodological knowledge, product knowledge, soft skills)?
   c. How are the people being trained evaluated before, during and after the learning process?
   d. What are the learning practices/activities employed during the learning process?
   e. In which levels does training take place (individual, group)?
   f. How is the learning process evaluated?
4. How does the onboarding process work?
   a. What are the scenarios wherein people are onboarded (single or group onboarding)?
   b. What aspects are considered when forming new or restructuring existing teams?
   c. What are the team-building activities performed?
   d. How are the teams evaluated?
   e. What is perceived as a mature team?
   f. What is perceived as a stable team?
   g. How is the onboarding process evaluated?

Figure A.1: Interview guide 1.
1. Please provide your background.
2. How does the recruiting process work?
   a. What are the characteristics that are mandatory for a candidate to be hired?
   b. What are the characteristics that are not appreciated in new candidates?
   c. Is the recruiting process evaluated?
3. How does the learning process work?
   a. What are the scenarios wherein developers need to be trained?
   b. What are the main learning needs (technical knowledge, methodological knowledge, product knowledge, soft skills)?
   c. How are the people being trained evaluated before, during and after the learning process?
   d. What are the learning practices/activities employed during the learning process?
   e. In which levels does training take place (individual, group)?
   f. How is the learning process evaluated?
4. How does the onboarding process work?
   a. What are the scenarios wherein people are on-boarded (single or group onboarding)?
   b. What aspects are considered when forming new or restructuring existing teams?
   c. What are the team-building activities performed?
   d. How are the teams evaluated?
   e. What is perceived as a mature team?
   f. What is perceived as a stable team?
   g. How is the onboarding process evaluated?

Figure A.2: Interview guide 2.
## BREBIAS QE REGRESSION MODEL DETAILS AND PLOTS

Table B.1: BREbias QE regression analysis: ANOVA.

<table>
<thead>
<tr>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>93.21</td>
<td>2</td>
<td>46.60</td>
<td>20.98</td>
</tr>
<tr>
<td>Residual</td>
<td>126.61</td>
<td>57</td>
<td>2.22</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>219.82</td>
<td>59</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table B.2: BREbias QE regression coefficients’ details.

| Unstd. Co- | Std. Error | Std. Co- | t     | Sig | Tolerance |
| Coeffs. B  |            | Coeffs. B|       |     |           |
| Constant   | -0.73      | 0.41      | -1.78 | 0.08 |           |
| Team Maturity | -0.56 | 0.20      | -0.28 | -2.78 | 0.007  | 0.996   |
| PC Size    | 1.22       | 0.20      | 0.61  | 6.01 | 0.000    | 0.996   |

Table B.3: BREbias QE regression models’ summary.

<table>
<thead>
<tr>
<th>Variables</th>
<th>R²</th>
<th>Adjusted R²</th>
<th>PC Size</th>
<th>PC Priority</th>
<th>Team Maturity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.42</td>
<td>0.39</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>0.42</td>
<td>0.40</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Figure B.1: Histogram of residuals.
Figure B.2: Normal PP Plot of regression standardized residuals.
Figure B.3: Scatter plot standardized residuals and predicted values.
**BREBIAS AE REGRESSION MODEL DETAILS AND PLOTS**

Table C.1: BREbias AE regression: ANOVA.

<table>
<thead>
<tr>
<th></th>
<th>Sum of Squares</th>
<th>df</th>
<th>Mean Square</th>
<th>F</th>
<th>Sig</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>16.36</td>
<td>2</td>
<td>8.18</td>
<td>6.79</td>
<td>0.002</td>
</tr>
<tr>
<td>Residual</td>
<td>68.69</td>
<td>57</td>
<td>1.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>85.05</td>
<td>59</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table C.2: BREbias AE regression coefficients’ details.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>-0.07</td>
<td></td>
<td>0.25</td>
<td></td>
<td>-0.27</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>PC Priority</td>
<td>-0.69</td>
<td>-0.29</td>
<td>0.31</td>
<td>-2.251</td>
<td>0.028</td>
<td>0.862</td>
<td></td>
</tr>
<tr>
<td>PC Size</td>
<td>0.57</td>
<td>0.45</td>
<td>0.16</td>
<td>3.54</td>
<td>0.001</td>
<td>0.862</td>
<td></td>
</tr>
</tbody>
</table>

Table C.3: BREbias AE regression models’ summary.

<table>
<thead>
<tr>
<th>Variables</th>
<th>R²</th>
<th>Adjusted R²</th>
<th>PC Size</th>
<th>PC Priority</th>
<th>Team Maturity</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.19</td>
<td>0.15</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>0.19</td>
<td>0.16</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Figure C.1: Histogram of residuals.
Figure C.2: Normal PP Plot of regression standardized residuals.
Figure C.3: Scatter plot standardized residuals and predicted values.
BIBLIOGRAPHY


CSCW’00, pages 319–328, New York, NY, USA, 2000. ACM. (Cited on pages 41 and 47.)


140 H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947. (Cited on pages 195 and 209.)


ABSTRACT

Background: Recruitment and onboarding of software developers are essential steps in software development undertakings. The need for adding new people is often associated with large-scale long-living projects and globally distributed projects. The former are challenging because they may contain large amounts of legacy (and often complex) code (legacy projects). The latter are challenging, because the inability to find sufficient resources in-house may lead to onboarding people at a distance, and often in many distinct sites. While onboarding is of great importance for companies, there is little research about the challenges and implications associated with onboarding software developers and teams in large-scale globally distributed projects with large amounts of legacy code. Furthermore, no study has proposed any systematic approaches to support the design of onboarding strategies and evaluation of onboarding results in the aforementioned context.

Objective: The aim of this thesis is two-fold: i) identify the challenges and implications associated with onboarding software developers and teams in large-scale globally distributed projects; and ii) propose solutions to support the design of onboarding strategies and evaluation of onboarding results in large-scale globally distributed legacy projects.

Method: In this thesis, we employed literature review, case study, and business process modeling. The main case investigated in this thesis is the development of a legacy telecommunication software product in Ericsson.

Results: The results show that the performance (productivity, autonomy, and lead time) of new developers/teams onboarded in remote locations in large-scale distributed legacy projects is much lower than the performance of mature teams. This suggests that new teams have a considerable performance gap to overcome. Furthermore, we learned that onboarding problems can be amplified by the following challenges: the complexity of the product and technology stack, distance to the main source of product knowledge, lack of team stability, training expectation misalignment, and lack of formalism and control over onboarding strategies employed in different sites of globally distributed projects. To help companies addressing the challenges we identified in this thesis, we propose a process to support the design of onboarding strategies and the evaluation of onboarding results.

Conclusions: The results show that scale, distribution and complex legacy code may make onboarding more difficult and demand longer periods of time for new developers and teams to achieve high performance. This means that onboarding in large-scale globally distributed legacy projects must be planned well ahead and companies must be prepared to provide extended periods of mentoring by expensive and scarce resources, such as software architects. Failure to foresee and plan such resources may result in effort estimates on one hand, and unavailability of mentors on another, if not planned in advance. The process put forward herein can help companies to deal with the aforementioned problems through more systematic, effective and repeatable onboarding strategies.