Architectural Knowledge Management in Global Software Development

Viktor Clerc

2011
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Jacquard

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormation about architectural knowledge.

MANAGING RISK

This research has been partially sponsored by DNV-CIBIT. Parts of this research have been performed in collaboration with DNV-CIBIT.

Promotiecommissie:
prof.dr. P. Avgeriou (University of Groningen)
prof.dr. S. Brinkkemper (Utrecht University)
dr. B. J. van den Hooff (VU University Amsterdam)
prof.dr. P.B.K. Kruchten (University of British Columbia)
dr. I. Richardson (Lero, University of Limerick)

NUR 982

SIKS Dissertation series No. 2011-40

Copyright © 2011, Viktor Clerc
All rights reserved unless otherwise stated

Published by Uitgeverij BOXPress, Oisterwijk
Printed on FSC-certified paper

Cover illustration by Mark van ’t Zet
Typeset in L\LaTeX by the author
Architectural Knowledge Management in Global Software Development

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de facultéit der Exacte Wetenschappen
op dinsdag 13 december 2011 om 11.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Viktor Clerc

geboren te Amsterdam
promotor: prof.dr. J.C. van Vliet
copromotor: dr. P. Lago
Table of Contents

Acknowledgments xxx

1 Introduction 1

1.1 Setting the Stage 2

1.1.1 Software architecture 2

1.1.2 Knowledge management 3

1.1.3 Architectural knowledge 4

1.2 Introducing Global Software Development 5

1.2.1 Definition of GSD 5

1.2.2 Challenges with GSD 6

1.2.3 Knowledge management in GSD 7

1.2.4 Benefits of GSD 7

1.2.5 Agile software development and GSD 8

1.2.6 Practices or solutions to achieve GSD benefits 9

1.2.7 Tool support for GSD 10

1.3 Research Context 11

1.4 Problem Statement 11

1.5 Research Questions 12

1.6 Research Methods and Studies 14

1.6.1 Research Methods 15

1.6.2 1 – Survey: How is architectural knowledge shared? 17

1.6.3 2 – Case study: Understanding practices for AKM in GSD 18

1.6.4 3 – Study: Identifying product practices for AKM in GSD 18

1.6.5 4 – Case study: Identifying practices for AKM in GSD 19

1.6.6 5 – Study: (Semantic) wiki support for AKM in GSD 20

1.6.7 6 – Case study: How is architectural knowledge managed 20

1.7 Publications 21

2 The Mindset of Architects 23

2.1 Introduction 23

2.2 Related Work 24

2.3 Research Design 25

2.3.1 Survey design 25

2.3.2 Selection of participants of the survey 26

2.3.3 Control of the survey 26

2.4 Findings 26

2.4.1 Demographic information 27

2.4.2 Architecture levels 27

2.4.3 Architectural roles 29

2.4.4 Clustering the use cases 30

2.4.5 Participants’ views on the use cases 32
## Table of Contents

2.5 Threats to Validity .................................................. 35  
2.6 Discussion and Conclusions ..................................... 36  
2.7 Future Work ......................................................... 38  

3 AKM Practices for Compliance in GSD ......................... 41  
3.1 Introduction ......................................................... 41  
3.2 Related Work ......................................................... 42  
3.3 Case Study Description ........................................... 43  
3.4 Performing the Assessment ....................................... 45  
3.4.1 Analysis of the population .................................. 46  
3.4.2 Conduct questionnaires ....................................... 49  
3.4.3 Conduct follow-up interviews ............................... 51  
3.4.4 Provide recommendations .................................... 53  
3.4.5 Classify architectural rules ................................. 54  
3.5 Results of the Case Study ......................................... 55  
3.6 Conclusions .......................................................... 56  

4 Reassessing the Need for Practices for Architectural Compliance ......................... 57  
4.1 Introduction ......................................................... 57  
4.2 Related Work ........................................................ 58  
4.3 Challenges in Global Software Development .................. 59  
4.3.1 Challenge 1: Time difference and geographical distance 60  
4.3.2 Challenge 2: Culture ............................................ 60  
4.3.3 Challenge 3: Team communication and collaboration .... 61  
4.3.4 Challenge 4: Work distribution .............................. 64  
4.4 Organizations’ Solutions to GSD Issues ....................... 66  
4.4.1 Organization A .................................................... 66  
4.4.2 Organization B .................................................... 68  
4.5 Conclusions .......................................................... 70  

5 Product Practices for AKM in GSD ......................... 73  
5.1 Introduction ......................................................... 73  
5.2 Related Work ........................................................ 74  
5.3 Context and Approach ............................................ 75  
5.3.1 Software quality evaluation framework .................... 75  
5.3.2 Research approach ............................................. 75  
5.3.3 Key elements of architectural knowledge ................... 76  
5.3.4 Architectural knowledge evaluation criteria ............... 76  
5.4 Results ............................................................... 77  
5.4.1 Characterization of the selected software development projects 77  
5.4.2 Evaluation results .............................................. 78  
5.5 Discussion of the Results ......................................... 80  
5.5.1 Limitations ....................................................... 81
# Table of Contents

## 6 Identifying Practices for AKM in GSD

6.1 Introduction ........................................ 83
6.2 Research Approach .................................. 84
   6.2.1 Literature research .............................. 84
   6.2.2 A pattern language for describing AKM practices 85
6.3 Architectural Knowledge Management Practices for GSD ...... 86
   6.3.1 Frequent interaction across sites ............. 86
   6.3.2 Cross-site delegation .......................... 87
   6.3.3 Face-to-face project kick-off meeting ........ 87
   6.3.4 Urgent request ................................ 88
   6.3.5 Collocated high-level architecture phase .... 89
   6.3.6 A clear organization structure with comm. responsibilities 89
   6.3.7 Have a repository for architecture artifacts ... 90
6.4 Validation ............................................ 91
6.5 Conclusions ......................................... 92

## 7 The Usefulness of AKM Practices in GSD

7.1 Introduction ......................................... 93
7.2 Related Work ........................................ 94
7.3 Research Question .................................. 95
7.4 Research Approach .................................. 96
   7.4.1 Organization context ........................... 96
   7.4.2 Approach ....................................... 96
7.5 Analysis and Results ................................ 98
   7.5.1 Demographic information ....................... 98
   7.5.2 Data preparation ................................ 99
   7.5.3 Perceived usefulness of AKM practices .... 99
7.6 Discussion of the Results ............................ 101
   7.6.1 Analysis of the number of sites ............... 101
   7.6.2 Analysis of AKM practices ..................... 103
   7.6.3 What’s special about three sites? ............ 103
   7.6.4 Limitations ................................... 105
7.7 Conclusions ......................................... 105

## 8 Supporting AKM in GSD

8.1 Introduction ......................................... 107
8.2 Research Question and Approach ........................ 109
   8.2.1 Research question ............................... 109
   8.2.2 Research approach .............................. 109
8.3 Wiki Functionalities .................................. 110
8.4 Implementing AKM Practices for GSD with Wiki Functionalities 111
   8.4.1 AKM practices for GSD .......................... 111
   8.4.2 Mapping AKM practices for GSD onto wiki functionalities 113
8.5 Implementing a Semantic Wiki ........................ 116
   8.5.1 Use cases implemented in a semantic wiki ... 116
Table of Contents

9 The Use of AKM Practices in Agile GSD
  9.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 125
  9.2 Research Question and Approach . . . . . . . . . . . . . . . . . . . . . . 126
    9.2.1 Research question . . . . . . . . . . . . . . . . . . . . . . . . 126
    9.2.2 Characterization of the organization . . . . . . . . . . . . . . . 127
    9.2.3 Research approach . . . . . . . . . . . . . . . . . . . . . . . . 129
  9.3 Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130
    9.3.1 Validation of AKM practices for GSD . . . . . . . . . . . . . . . 130
    9.3.2 Additional AKM practices . . . . . . . . . . . . . . . . . . . . . 134
    9.3.3 Major findings . . . . . . . . . . . . . . . . . . . . . . . . . . 134
  9.4 Threats to Validity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136
  9.5 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136

10 Conclusions
  10.1 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139
    10.1.1 How is architectural knowledge used? (RQ 1) . . . . . . . . . . . 139
    10.1.2 What are practices for AKM in GSD? (RQ 2) . . . . . . . . . . . 140
    10.1.3 How can AKM (practices) in GSD be supported? (RQ 3) . . . . . . 141
    10.1.4 Which practices for AKM in GSD are used? (RQ 4) . . . . . . . . . 141
    10.1.5 How can AK be managed in a GSD environment? . . . . . . . . . 142
  10.2 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
    10.2.1 Linking to other adjacent fields . . . . . . . . . . . . . . . . . . 142
    10.2.2 Reflections on our contributions . . . . . . . . . . . . . . . . . . 143
    10.2.3 A glance at the future . . . . . . . . . . . . . . . . . . . . . . . 144

Appendix

A Overview of the Perceived Usefulness of AKM Practices . . . . . . . . . . . 147

Samenvatting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 151

SIKS Dissertation Series . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155

Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 167
### List of Figures

1.1 Core model of architectural knowledge ........................................ 5
1.2 Research questions ........................................................................ 14

2.1 Overview of the architect’s mindset ................................................. 36

3.1 Projects versus subsystems ............................................................. 44
3.2 Overview of assessment activities .................................................. 46
3.3 Usage of archnotes per site ............................................................. 47
3.4 Number of hits per archnote per development site ....................... 48

4.1 Overview of challenges and their interrelations ............................... 60

7.1 Overview of the usefulness of AKM practices .................................. 102

8.1 Scenario 1 through the semantic query interface in *SE-Wiki* ............. 119
8.2 Scenario 2 through an in-line semantic query in *SE-Wiki* ............... 120
8.3 Scenario 3 through comparison in *SE-Wiki* .................................. 121

9.1 Typical project team structure at Organization E ............................ 128
9.2 Use of the “frequent interaction across sites” practice ..................... 131

A.1 Frequent interaction across sites .................................................... 147
A.2 Cross-site delegation ..................................................................... 148
A.3 Face-to-face project kick-off meeting ............................................ 148
A.4 Urgent request ............................................................................... 148
A.5 Have a clear organization structure with comm. responsibilities ...... 149
A.6 Travel to other project locations ................................................... 149
A.7 Know who is who across the project (directory) .............................. 149
A.8 Have a single repository for architecture artifacts ........................... 150
A.9 Have a shared infrastructure for work products and source code ....... 150
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Overview of studies and research methods</td>
<td>15</td>
</tr>
<tr>
<td>2.1</td>
<td>The percentage of time spent by all respondents on architecture levels</td>
<td>28</td>
</tr>
<tr>
<td>2.2</td>
<td>Clusters of architecture levels</td>
<td>29</td>
</tr>
<tr>
<td>2.3</td>
<td>Clusters of architectural roles</td>
<td>30</td>
</tr>
<tr>
<td>2.4</td>
<td>Use cases for architectural knowledge</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Importance of use case clusters</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Practices for knowledge sharing</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Most important archnotes</td>
<td>51</td>
</tr>
<tr>
<td>3.3</td>
<td>Opinion on the architecture team</td>
<td>52</td>
</tr>
<tr>
<td>3.4</td>
<td>Problems that prevent securing architectural rules</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of organizations’ solutions to GSD issues</td>
<td>71</td>
</tr>
<tr>
<td>5.1</td>
<td>Key elements of architectural knowledge</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>Architectural knowledge evaluation criteria</td>
<td>77</td>
</tr>
<tr>
<td>5.3</td>
<td>Scale for evaluation criteria implementation</td>
<td>78</td>
</tr>
<tr>
<td>5.4</td>
<td>Ordered results of assessment analysis</td>
<td>79</td>
</tr>
<tr>
<td>6.1</td>
<td>Pattern language</td>
<td>85</td>
</tr>
<tr>
<td>7.1</td>
<td>Survey excerpt with those questions relevant to this research</td>
<td>98</td>
</tr>
<tr>
<td>7.2</td>
<td>Number of responses per number of sites</td>
<td>99</td>
</tr>
<tr>
<td>8.1</td>
<td>Relating AKM practices for GSD and wiki functionalities</td>
<td>114</td>
</tr>
<tr>
<td>9.1</td>
<td>Practices for AKM in GSD</td>
<td>127</td>
</tr>
<tr>
<td>9.2</td>
<td>Overview of results per site</td>
<td>131</td>
</tr>
</tbody>
</table>
Acknowledgments

While employed at the Software Engineering Research Centre (SERC) in Utrecht, the Netherlands, one of my colleagues once said: “Having obtained a Ph.D. prior to starting employment at SERC, there is a single thing I would never recommend to anyone: pursuing a part-time Ph.D.” Well, this book is the tangible evidence that it in fact can be done, albeit not without the great help of so many people supporting me.

First of all, I sincerely would like to thank my supervisors Hans van Vliet and Patricia Lago. Hans, for guiding me along the complete journey: our initial discussions late 2004, setting up the various necessary formalities, and for his continuous effort in making me “focus” and keeping me in line. Now I see that’s what it is about in performing research – I really thank Hans for offering me this opportunity, and for the opportunity to finish my research when the research project formally had ended. With Patricia, I obviously share the love for Italy and the game of baseball. Even more important, I greatly value the continuous discussions on the set up of research experiments, identification of new angles to address observations, and, again, showing me how to keep “focus”.

In performing this research, I had the opportunity to collaborate with so many bright colleagues within the GRIFFIN and Stephenson research projects. At the University of Groningen: Dieter K. Hammer, Paris Avgeriou, Anton Jansen, Peng Liang, and Jan S. van der Ven. At the VU University, the list of colleagues is even longer: first of all, a thanks to Rik Farenhorst and Remco C. de Boer for being perfect colleagues in the GRIFFIN-team; especially at the times I was physically present at the university, but also during off hours. Specifically, Remco’s sharpness in the various debates we had and Rik’s cooperativeness contributed to the results of my research. Next, the continuous evolution of the research group at the VU brought many fruitful insights: I thank Qing Gu, Maryam Razavian, and Damian Tamburri for this. Together with Antony Tang and Christina Manteli I enjoyed to carry out my final case study in the Stephenson project. Joost Schalken and Rahul Premraj provided indispensable statistical advice in some of the case studies. Finally, I am thankful for the support of Master’s students that contributed to parts of this research: Ibrahim Düz and Edwin de Vries.

The research reported in this thesis is the result of various case studies at several industrial partners. It has been a pleasure in collaborating with these partners and in particular Robert Deckers, Tim Trew, Heleen Hollenberg, Rob van Twist, Guido Schoonheim, Eltjo Poort, and Hub van de Laar.

Though research contexts never are fully stable, this certainly was not the case for my employer’s context. After my professional career had started at SERC, it continued at CIBIT, which later became DNV-CIBIT, and even more recently inspearit; “never a dull moment”, one might say. During my professional career, it was possible for me to gain new insights and to grow and develop myself as a professional. While collaborating with Dutch and foreign colleagues on various significant projects, it was a pleasure observing the interest for my research expressed by them.

Needless to say, I greatly appreciate the opportunity of pursuing my Ph.D. while continuing my work as IT consultant and trainer. This purely win-win situation was initially discussed with Ernst van Aagten and Gert Florijn, later formalized thanks to Johan den Biggelaar, and ultimately, while approaching the final stages, supported by Yann Ha-
Acknowledgments

Support in off-work hours was to a great extent provided by my friends. Special thanks go to Jasper Videler; thanks for being a friend, playing ball, and literally whetting and developing my culinary appetite; I will surely spend more time at your place! In addition, I thank the guys and girls from AMVJ for the perspective they brought while playing ball, sharing drinks, and enjoying numerous (non-)sport trips.

My parents Hans and Conny, brother René, and other family always showed me that it is important to fully pursue ambition, starting day one; this advice has paid off and continues to guide me in life.

There is one person who never could have foreseen what the decision to step into this research actually means. Having met me when I just started this journey, you supported and loved me in all that I do; late at night, early in the morning, or even abroad. Rianne, thanks for your patience and support: I can’t wait to step into the future with you.

Amstelveen, October 2011
Viktor Clerc
Introduction

The use of software in our daily lives plays an increasingly important role. Software enables us to communicate across geographical and organizational borders: it allows us to stay in contact with loved ones that enjoy their holidays at the other side of the world and facilitates multi-national organizations to collaborate (cost-)effectively. These developments increase the need for software to satisfy the needs of its ever increasing demand of use.

Software engineering is the discipline that covers a structured approach of developing (large-scale) software systems; it is “the application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software” ([IEEE, 1990]). Although the notion of software engineering indeed arose from the comparison with engineers and civil architects, one of the main differences between software engineering and civil engineering includes the actual physical effort put into civil craftsmanship (construction of a building) versus software craftsmanship (building software, e.g., via generation). Although the discipline of software engineering is relatively young as compared to civil engineering, several practices from civil engineering have been successfully translated for use in software engineering, e.g., the importance of architecting activities and the definition and use of design patterns ([Alexander et al., 1977]).

Traditionally, software development occurred using a waterfall-like development process. In this process, the requirements are fixed and the software design is built and completed based upon those requirements. Next, the implementation commences and when implementation is finished, the testing phase is started. The waterfall development process does only to a limited extent cater for incorporation of the experiences collected in each of the analysis, design, or implementation activities, see ([Royce, 1970]). To overcome these difficulties and address the explicit need to cater for changing requirements during the software development lifecycle, more incremental and iterative software development processes matured. [Larman and Basili, 2003] provide a good overview by describing a brief history of iterative and incremental development processes.

All in all, software engineering has matured over the last decades. Since the term was first coined at the NATO conference on Software Engineering ([Naur and Randell, 1968]), several improvements have been made – yet, several authors still refer to the software crisis in that software engineering in fact boils down to “(...) handcrafting from
raw programming languages by artisans using techniques they neither measure nor are able to repeat consistently” (Gibbs, 1994; The Standish Group International, 1994); others, however, place some question marks at the use of the term crisis, consider e.g., (Glass, 2006). One may conjecture that, because of the ongoing globalization which allows for continuous learning, more and more best practices are identified and incorporated in software engineering methods. New insights are gained on almost a daily basis – these insights require more fundamental and empirical research to provide a scientific basis for its adequate and successful application. This is what has been done in this thesis: we join two recent disciplines and identify how mainly empirically obtained research insights can contribute in increasing the success of each of them. The first discipline concerns the management of architectural knowledge; the integrated representation of the architecture of a software system including e.g., design decisions (see §1.1.3 for a full definition). The second discipline concerns global software development, one of the most frequently chosen solutions by organizations to lower development cost (Carmel and Agarwal, 2001) or speed up development time (Carmel, 1999).

In this chapter, we lay the foundation for the remainder of this thesis. We set the stage by providing a summary of the key advances in the area of software architecture, knowledge management, and architectural knowledge in §1.1. Next, we provide an extensive introduction into global software development in §1.2. We present the context in which we performed the research in §1.3. The problem statement for the research is given in §1.4. In §1.5 we present our research questions. Next, §1.6 describes the research methods used. Finally, in §1.7 we list the publications upon which this thesis is based.

1.1 Setting the Stage

1.1.1 Software architecture

One of the more recent advances of software engineering includes the subdiscipline of software architecture. Software architecture traditionally is regarded as a combination of elements, form, and rationale (Perry and Wolf, 1992). As Perry and Wolf describe, this comprises processing, data, or connecting elements, the properties and relationships among these elements, and the underlying basis for the architecture (e.g., constraints). Kruchten (1995) provides his experiences in describing a software architecture according to a view model consisting of several predefined views, each addressing specific concern, and a scenario view, allowing for illustration according to the architecture by means of a series of use cases.

A further elaboration of the notion of view model as provided by Kruchten is given in the IEEE 1471 recommended practice (IEEE, 2000). According to the standard, each software-intensive system has an architecture that can be described according to predefined viewpoints. These viewpoints are designed to accommodate for the concerns of designated stakeholders. One of the main advantages of view-based architectural descriptions is that it enables early interaction with stakeholders. In addition, its basis for a work breakdown structure and the early assessment of quality attributes is discussed.
1.1. Setting the Stage

in (Bass et al., 2003; Clements et al., 2003).

More recently, viewing the architecture of a software system as a set of design decisions has experienced an increase of interest (Bosch, 2004; Kruchten et al., 2006; van Vliet, 2008; Jansen, 2008; Farenhorst and de Boer, 2009). This new paradigm initiated a new research effort in the area of software architecture, namely how the architecture can be described effectively by means of these decisions, and how reasoning over the body of architectural knowledge can be further enhanced and supported. It has led to e.g., a decision view in software architecture (Dueñas and Capilla, 2005; Kruchten et al., 2009) and the definition of a virtual community for architectural knowledge sharing (Lago et al., 2010).

During recent years, the IEEE 1471 recommended practice has been superseded by ISO standard 42010 (ISO/IEC, 2011). This ISO standard even further incorporates the ideas of describing a software architecture as a set of design decisions by explicitly including the architecture rationale and architecture decisions as architectural description elements – this shows the wider acceptance of these rather new developments within the software architecture community.

1.1.2 Knowledge management

The discipline of knowledge management makes a clear distinction between two types of knowledge: tacit knowledge and explicit knowledge. Tacit knowledge pertains to the kind of knowledge one possesses but cannot easily express. Explicit knowledge is knowledge that can be expressed in various forms, such as conversations, electronic communication, or writing (Nonaka and Takeuchi, 1995); the majority of the knowledge remains tacit because no explicit actions are undertaken to codify the knowledge; if this is the case, the risk of key individuals leaving the organization (and the knowledge that walks out the door with them) increases further.

Following the insights from Nonaka and Takeuchi, several strategies for knowledge management can be distinguished. These strategies include a codification strategy towards knowledge management and a personalization strategy towards knowledge management. Hansen et al. (1999) define the codification strategy as aimed at systematically storing knowledge in a repository, structured or unstructured, so that it becomes available to people in the company. A personalization strategy promotes interaction between knowledge workers; knowledge is kept by its creator and not the knowledge itself but rather information about knowledge sources is captured, to ensure that people know who knows what. Desouza and Evaristo (2004) conclude that when knowledge is to be shared in distributed projects, a hybrid approach focusing on both personalization and codification is favourable.

We observe that knowledge management practices further strengthen the discipline of software engineering. Rus and Lindvall (2002) provide a good overview of the recent developments in software engineering for sharing documented knowledge independently of time and location. Some of these practices may include e.g., boundary spanning. Boundary spanning is emphasized by the literature as relying on individuals (i.e., the boundary spanners) to facilitate the sharing of expertise by linking two or more groups of people separated by location, hierarchy, or function (Cross and Parker, 2004).
1. Introduction

A more recent report on a study on practices for boundary spanning (Levina and Vaast, 2005a) has researched the balance between collaborative boundary spanning practices versus transactive boundary spanning practices (Kogut and Zander, 1992). As put forward by Kogut and Zander, a delicate balance lies between reliance on interpersonal collaborative practices versus practices that support the codification of knowledge. This notion has been independently researched and described in (Nonaka and Takeuchi, 1995; Hansen et al., 1999).

When the individual memory systems and communications of e.g., boundary spanners (labeled transactions) are combined, this results in a transactive memory system. Hence, a transactive memory system is a set of individual memory systems in combination with the communication that takes place between individuals (Wegner, 1986). Oshri et al. (2008) studied the role of transactive memory in knowledge transfer between globally distributed teams. Oshri et al. found that transactive memory supports the the notion of who knows what across on-site and off-shore teams despite the challenges associated with globally distributed teams.

1.1.3 Architectural knowledge

As described earlier, the management of knowledge pertaining to the software architecture becomes increasingly important within the discipline of software architecture (Bosch, 2004; van Vliet, 2008). Management of architectural knowledge helps to prevent design decisions to remain tacit which helps to reduce the high cost of change because of knowledge vaporization (Jansen et al., 2008).

De Boer and Farenhorst (2008) have performed research to define what architectural knowledge exactly is. Using a systematic literature review, de Boer and Farenhorst show that many authors do not provide a concrete definition of what architectural knowledge entails. Yet, these authors seem to agree that knowledge spans from problem domain through decision making to solution.

We define architectural knowledge as the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context or environment. As such, architectural knowledge includes both functional and technical architectural knowledge. The management of architectural knowledge covers all activities to create, store, retrieve, and maintain architectural knowledge. Hence, architectural knowledge management may be supported by practices and (automated) support tools.

For further reasoning about architectural knowledge, it is important to prevent possible terminology differences that may exist at various organizations, including those where we performed case studies during our research, see (de Boer and Farenhorst, 2008). To enable further reasoning, we have developed a core model of architectural knowledge. The core model has been devised to cover all concepts necessary to define architectural knowledge (de Boer et al., 2007): as such, it may help in reasoning about and codifying essential architectural knowledge. With the core model, constructing an Architectural Design can be seen as a decision-making process. Decision-making essentially boils down to proposing and Ranking Alternatives to address a specific Concern of a Stakeholder, and eventually results in making a decision (see the lower part
1.2 Introducing Global Software Development

in Fig. 1.1. Furthermore, the core model supports in describing architectural knowledge using a Language (e.g., an ADL, or just plain English) to capture the Architectural Design in an Artifact (again, see Fig. 1.1).

Figure 1.1: Core model of architectural knowledge (from (de Boer et al., 2007))

1.2 Introducing Global Software Development

1.2.1 Definition of global software development

Global software development (GSD) is software development that uses teams from multiple geographic locations (Sangwan et al., 2006). The locations of these teams are commonly referred to as sites or development sites. When comparing GSD to traditional, collocated software development, the most striking difference is the geographical spread of the development locations. In spite of this, research by Allen shows that the impact of geographical spread may occur more often than initially thought of; engineers do not communicate more frequently when they are 30 meters or more apart as compared to many miles (Allen, 1977). Nevertheless, challenges that occur because of geographical
1. Introduction

spread of development sites may be increased by several additional factors that play a significant role in GSD. These additional factors relate to cultural diversity and time zone difference. These topics will be elaborated upon later in this chapter.

Overviews and general experience reports on GSD have regularly appeared in IEEE Software (Herbsleb and Moitra, 2001; Damian, 2007; Carmel and Agarwal, 2001). More recently, the ICSE workshop series on GSD have evolved into the International Conference on Global Software Engineering (ICGSE) series. Several workshops on software engineering disciplines furthermore receive a ‘global’ focus, such as requirements engineering (GREW’07), knowledge management (KNOWING’09 and ’10), and architectural knowledge management (SHARK and AGSE workshop series).

1.2.2 Challenges with global software development

When spreading development activities across different sites, several challenges are posed on the assumed benefits of global software development. Agerfalk et al. (2005) provide a useable framework to consider the opportunities and challenges or threats involved in GSD. Agerfalk et al. have performed an extensive literature study and identified the following three processes that are fundamental for GSD:

1. Communication – “The exchange of complete and unambiguous information – that is, the sender and receiver can reach a common understanding”, according to Carmel and Agarwal (2001). The distance involved in GSD takes the opportunity for face-to-face exchange away, which makes communication in itself even more important.

2. Coordination – “The act of integrating each task with each organisational unit, so the unit contributes to the overall objective” (Carmel and Agarwal, 2001). Coordination between two parties makes these parties dependent of each other. In GSD, this dependency is burdened because of different time zones and cultural differences.

3. Control – “The process of adhering to goals, policies, standards, or quality levels” (Carmel, 1999). Control focuses on the management reporting mechanisms and hence relates to project management.

Holmström et al. (2006) built upon the framework provided by Agerfalk et al. by further elaborating on the pivotal concept of distance in GSD as initially mentioned by Carmel and Agarwal (2001): temporal distance refers to the limited amount of overlap in working hours (e.g., due to time zone difference or shifting working hours) and the limited possibility to synchronize work across various sites. Socio-cultural distance refers to the different cultures that cooperate in GSD – this is also reflected in linguistic differences. Socio-cultural distance has been extensively analyzed by Hofstede (2001). Hofstede has conducted several extensive studies and identified a theory on cultural dimensions, which shows how national and regional cultural groups influence the behavior of societies and organizations. Hofstede’s research provides insight into other
cultures so that one can be more effective when interacting with people in other countries. Geographical distance, finally, focuses not on the distance itself but on the effort needed for relocation, i.e., travelling. Low geographical distance typically offers more possibilities for collocated, inter-team working.

1.2.3 Knowledge management in global software development

Without effective information and knowledge sharing mechanisms, it is difficult to exploit the benefits of global software development (Prikладnicki et al., 2003). However, applying knowledge sharing mechanisms poses several challenges to GSD itself (Desouza et al., 2006). In GSD, expertise and best practices are collected and reside at multiple locations (possibly across organizations, in the case of outsourcing). Consequently, difficulties exist in locating and/or accessing relevant knowledge due to the distance innate to GSD. Moreover, coordinating and synthesizing this knowledge poses challenges: organizations don’t use relevant knowledge harvested from past experiences.

Different strategies towards the management of knowledge in GSD have been explored in the literature. As opposed to implementing one central knowledge structure in which all knowledge is kept and managed, Desouza et al. (2006) propose a hybrid knowledge strategy which is focused on reusing global knowledge and connecting knowledge sources for local (i.e., project-specific) knowledge. The structure for the global knowledge would allow for the use of popular knowledge and provide an index to project-specific knowledge, hence providing coordination across project work. As such, the models proposed by Desouza et al. (2006) build upon the codification strategy and personalization strategy towards knowledge management as initially described by Hansen et al. (1999).

1.2.4 Benefits of global software development

Up until recently, research has primarily strived to obtain a good understanding of the challenges involved in GSD. Obviously, assumed benefits exist that organizations strive to achieve when starting to develop software on a global scale. A survey of recent literature reports the following benefits (Ó Conchuir et al., 2009):

Reduced development costs due to salary savings (Carmel and Agarwal, 2001) – Typically, salaries are orders of magnitudes lower in off shore countries such as India or China, or near-shore locations such as that may be found in Eastern Europe.

Access to a larger (skilled) labor pool (Carmel and Agarwal, 2001) – For demographic reasons, the number of skilled software engineering practitioners is higher in e.g., India or China.

‘Follow-the-sun’ development leveraging time-zone effectiveness (Carmel, 1999) – Although organizations primarily choose for GSD for the assumed cost advantage and access to a larger labor pool, the real benefit of GSD lies in the fact that software development can occur 24x7 by handing over software development tasks (efficiently) to sites in appropriate time zones (Carmel and Agarwal, 2001). Visser and van Solingen (2009)
1. Introduction

offers additional guidance on how to select the right sites to fully utilize the benefits of GSD using an implemented routing model. Experience reports (Ó Conchuir et al., 2009) suggest a shift in working hours to get more overlap in working hours. On the other hand, this shift reduces the possibility for getting the most out of ‘follow-the-sun’ development).

Greater innovation and transfer of best practices (Damian et al., 2003) – As a direct consequence of access to a larger (skilled) labor pool, organizations that employ a lot of practitioners have the possibility to explore various (technological) innovations and share best practices as these are collected in their software development projects.

Closer proximity to markets and customers (Grinter et al., 1999; Herbsleb et al., 2000) – With upcoming (emerging) markets in Middle-East and Asia, the labor pool available in these countries (e.g., Eastern European countries, India, China) has a closer proximity which can result in alleviating the geographical and time zone distance.

Agerfalk et al. (2008) provide a list of several more unknown organizational, team, and process/task benefits, e.g., shared best practices, improved resource allocation, reduced coordination cost, increased team autonomy, improved documentation, and clearly defined processes. The study of Ó Conchuir et al. included these benefits in a study on to determine to what extent the benefits are realized in practice. The results show that most of the assumed benefits are realized only partially, if at all. Main reasons for not reaching the assumed benefits are the following (Ó Conchuir et al., 2009):

- Significant overhead in communication, coordination and control needed.
- Only less complex, less mission-critical tasks, and testing activities are performed using the ‘follow-the-sun’ approach.
- Although a larger pool of skilled developers is available, socio-cultural problems such as linguistic problems (vocabulary and interpretation) (Holmström et al., 2006) do arise.
- Employees who feel threatened by the larger labor pool becoming available are unlikely to share best practices or knowledge.

1.2.5 Agile software development and global software development

Agile software development covers a broad range of software development methodologies that are aimed at delivering working software soon, interacting with customers, and responding to change (Beck, 2001). Agile software development methodologies promote short development cycles and a focus more on working software over comprehensive documentation.

Different opinions exist on whether or not agile software development methodologies can flourish in a global software development environment. Although increased geographical distance reduces the ability for coffee talks and other ways of informal
1.2. Introducing Global Software Development

communication, nowadays communication infrastructure is present (see [1.2.7]) that reduces distance without the need for physical travelling \[1\]. Often, a higher need for documentation in GSD as compared to collocated software development is denoted. On the other hand, in order to make GSD a success, conscious adoption of agile practices, which have a lesser focus on documentation, can be helpful – as long as the adoption process itself is open for change (Ramesh et al., 2006; Sutherland et al., 2008).

David Parnas makes several remarks on the disciplines of GSD and agility in the sidebar in (Ågerfalk and Fitzgerald, 2006). Parnas states that the underlying problems in GSD boil down to communication problems – between users and requirements engineers, between architects and developers, and between developers themselves. Global software development only worsened these problems. Solutions to the problems do not lie in producing more and more documentation of some kind. Parnas: “The real grand challenge is not to find ways to avoid documenting, but to find ways to produce useful documents – documents that take time but save more time. We will find that real agility comes from good design that is well documented in precise, lean documentation.” (Ågerfalk and Fitzgerald, 2006).

1.2.6 Practices or solutions to achieve global software development benefits

When elaborating on the concept of distance and the challenges associated with it, solutions to the challenges in global software development may be found in practices that help to overcome the distance. Following Carmel and Agarwal (2001), distance negatively affects communication, which in turn hampers coordination effectiveness. Conway’s Law (Conway, 1968), which states the structure of an organization is mirrored in the structure of the software produced by that organization, calls for a clear modularization of the work to be performed. Various researchers have reported on reduction of coordination by means of a clear modularization of work to allow for localized (design) decisions and hence prevent conflicts (Herbsleb and Grinter, 1999b). To go even further, Carmel (1999) states that the architecture of a software system should be the driving force for modularizing and dividing work, instead of the other way around. More recently, collaboration tool support can be utilized to cover the best of both worlds: communicating effectively to divide or (re-)distribute work.

Solutions for the challenges in GSD are typically provided in terms of communication and coordination strategies; overviews are provided in (Herbsleb and Grinter, 1999a; Carmel and Agarwal, 2001; Lanubile et al., 2003; Herbsleb et al., 2005). These authors, amongst others, suggest the following tactics to overcome distance:

Reduce intensive collaboration – By a clear division of work, the need for intensive cross-module collaboration and communication is reduced.

Reduce cultural distance – Cultural distance manifests itself in two forms: difference in organizational culture (the norms and values of the organizational unit) and national

\[1\] On the other hand, not travelling at all is not a solution either. Rather, travelling in the early phases of a project helps to build trust and establishes team commitment (Corry et al., 2006).
1. Introduction

culture (ethnicity, language, and political boundaries). Solutions to reduce cultural distance include:

- Introduce a **bridgehead** by having a fixed percentage of the work be done on-site and thus not off-shoring 100% of the work.

- Open **internal (wholly owned)** foreign software development centers instead of outsourcing or subcontracting the work to different organizations. In this way, one manages to reduce organizational distance and is able to train the employees in corporate methodologies.

- Appoint a **liaison** who is travelling back and forth between the stakeholder sites. This liaison can fulfill an architect’s role as mentioned by (Corry et al., 2006) or could even be a **cultural** liaison (e.g., an ex-patriate) as (Carmel and Agarwal, 2001) suggests.

- Focus on improving the **language** used by the employees involved in GSD, by e.g., offering language courses.

**Reduce temporal distance** – Although different asynchronous communication mechanisms exist (e.g., email, online discussion groups), advantages are brought by **synchronous** communication mechanisms such as audio- or videoconferencing: smaller problems can be solved before they evolve into bigger ones. Although synchronous communication reduces temporal distance, it also prevents the benefits from ‘follow-the-sun’ work to be fully harvested. Moreover, the success of synchronous communication can be influenced by a difference in national culture; practitioners may be less or more eager to engage in synchronous communication because of e.g., language barriers.

Balaji and Ahuja (2005) have performed research that shows that a knowledge integration strategy can help to overcome the challenge of dispersed knowledge as highlighted by (Desouza et al., 2006). Their research shows that adhering to a knowledge integration strategy rather than choosing for one global knowledge management system, has a positive impact on overcoming challenges in GSD.

Casey and Richardson (2009) have developed a generic implementation model that supports organizations in adhering to a practical and systematic approach to address the key activities, infrastructure, and support required to facilitate effective GSD. This is a model in which the aforementioned practices may be placed. Richardson et al. (2010) propose a software process (inspired by the CMMI) to support the change for the project management discipline by proposing several practices for organizations implementing GSD.

1.2.7 **Tool support for global software development**

Recently, different tools have been developed to address the issues involved in global software development and to integrate several of the solutions described earlier. In this section, we summarize some of the major developments. This section is primarily based on literature from the field (Lanubile et al., 2010; Cataldo et al., 2009).
Collaboration tooling focuses on supporting collaboration throughout the software engineering lifecycle. Version control systems (e.g., Concurrent Versions System, Subversion) allow for over-the-web management of evolution of source code and artifacts. Issue tracking systems allow for e.g., distribution of (corrective maintenance) tasks to different sites. Other tools allow for collaborative modeling (and not just sharing the results of collocated modeling) using UML or other formal or semiformal languages. Finally, communication tools such as email, mailing lists, on-line meeting facilities, groupware tools and more recently Web 2.0 solutions such as (micro-) blogs and wikis have proven their value to software developers at multiple sites, see e.g., (Damian et al., 2009).

Certain tools support only a (set of) software engineering life-cycle activities: project management, requirements engineering, architecture and design (Capilla et al., 2007; Cataldo et al., 2009), and testing. Yet, no current tool supports all activities necessary for GSD. Lanubile et al. (2010) conclude by stating that “users must (…) prioritize their collaboration needs and the tools to support them” instead of the other way around.

1.3 Research Context

The research presented in this thesis has been performed in participation with two Dutch research projects: GRIFFIN and Stephenson.

The GRIFFIN (a GRId For inFormatIoN about architectural knowledge) project was a four year multi-partner research project. The goal of GRIFFIN was to identify tools, methods, and techniques for managing architectural knowledge. The project involved two universities (VU University Amsterdam and the University of Groningen) and several industrial organizations. The organizations involved range from SMEs to multinationals, and from scientific institutes to IT service providers. In the GRIFFIN research, we had the opportunity to collaborate with two of these partners.

The Stephenson project is a joint project between VU University Amsterdam and an industrial organization. In the project, research in the area of sharing architecture knowledge in a multi-site context was performed within the context of software product line development.

The GRIFFIN project was sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406. The Stephenson project was sponsored by the Dutch “Regeling Kenniswerkers”, project KWR09164.

1.4 Problem Statement

As discussed (1.1.3) the management of knowledge plays an increasingly important role in the discipline of software engineering. One of the most important types of knowledge is knowledge pertaining to the architecture of the system being built. By sharing this architectural knowledge, challenges such as design erosion, high maintenance costs, and lack of information or documentation can be further reduced (Jansen, 2008).
When compared with collocated software development, global software development poses additional challenges. These challenges are related to temporal, geographical, and socio-cultural distance, as set forth by (Ågerfalk et al., 2005; Holmström et al., 2006). Communication and coordination between various development sites is hampered because of the time difference (limited overlap in time available to synchronize work), socio-cultural distance, and geographical distance. We have elaborated on these challenges in §1.2.2.

The combination of aforementioned two recent developments (the management of architectural knowledge and GSD) may appear to be fruitful; the challenges and issues involved in GSD may be addressed using architectural knowledge management and, conversely, the expected benefits of GSD may be further leveraged using architectural knowledge management techniques. On the other hand, specific architectural knowledge management techniques may be necessary in sharing architectural knowledge across time zones and geographical borders. Likewise, the form in which architectural knowledge is shared should be chosen deliberately to overcome cultural barriers, such as differences in language. In conclusion, we currently lack the insight into how architectural knowledge can be managed in GSD.

1.5 Research Questions

The problem statement as put forward in the previous section provides the motivation for this research: understanding how architectural knowledge management and global software development can co-exist, and how architectural knowledge management can further support the benefits from GSD and address challenges and issues involved with GSD. Hence, we formulate our central research question as follows:

**RQ How can architectural knowledge be managed in a global software development environment?**

Based on the definition of architectural knowledge as provided in §1.1.3, we have further elaborated the use of architectural knowledge within the GRIFFIN project. As a first step towards understanding how architectural knowledge is managed, we pose the research question below. This research question is answered in Chapter 2 of this thesis.

**RQ-1 How is architectural knowledge used?**

For addressing the central research question, we are interested in identifying practices for the management of architectural knowledge in GSD. This directly leads to our second research question.

**RQ-2 What are practices for managing architectural knowledge in a global software development environment?**

The second research question serves as the main contribution of the work presented in this thesis. In our research, we have applied a breakdown of RQ₂ by identifying several distinct areas of research. In performing this research, we collected the results with which we are able to answer RQ₂. We describe each of these research areas in
turn. As such, the breakdown as provided below provides us with step by step results needed to answer RQ-2.

- First, we identified what practices a typical software development organization involved in GSD uses. Next, we compared these typical practices with the practices used by another organization to further obtain confidence in the necessity or usefulness of these practices. This first study provided us with the insight that primarily architectural knowledge management practices related to the development process were used in the organizations studied. This part of RQ-2 is answered in Chapters 3 and 4 of this thesis.

- While our understanding of the management of architectural knowledge matured, we used the insight obtained to choose an alternative perspective towards the management of architectural knowledge in GSD. This alternative perspective focuses on managing architectural knowledge in GSD by capturing the architectural knowledge in the software product. With software product, we refer to the software architecture, the software itself, and its documentation, following (ISO/IEC, 2000). The software product can be a (possible intermediate) tailor-made solution, since we do not focus on commercial-off-the-shelf components in our research. Hence, we posed an additional question to identify whether or not the quality of software products developed using GSD, can be improved by architectural knowledge stored in the products. This question is answered in Chapter 5.

- As we have learned during the course of our research, the requirements engineering discipline is a well-discussed example of a discipline that becomes challenging in GSD (Damian, 2007; Hsieh, 2006). Furthermore, several solutions have been proposed and validated in the requirements engineering discipline. The similarity between the requirements and architecture and, more specifically, requirements and (architectural) design decisions has been pointed out by (van Vliet, 2008; de Boer and van Vliet, 2009). We have chosen the requirements perspective to identify whether architectural knowledge management can leverage the solutions that are available from the requirements engineering domain to address the challenges in GSD. As such, we identify proposed solutions using this alternative requirements perspective. We validated the usefulness of these proposed solutions at GSD projects that ran an industrial partner in our research and specifically looked into the number of sites involved in these GSD projects. The proposed solutions are described in Chapters 6 and 7.

Practitioners involved in managing architectural knowledge in GSD are only motivated to do so if they are supported adequately. As described in §1.2.6, several supporting solutions are proposed but not yet aimed at managing architectural knowledge. Hence, we identify possible supporting instruments (tools) for the practices that identified in RQ-2. The answer to this question is provided in Chapter 8.

**RQ-3** How can architectural knowledge management (practices) in global software development be supported (by tools)?
To fully understand the applicability of the practices for architectural knowledge management in GSD, we investigate which of our defined practices (see RQ-2) are actually used in GSD practice. The answer to this question is provided in Chapter 9.

**RQ-4** Which practices for managing architectural knowledge are used in a global software development environment?

Fig. 1.2 illustrates the research questions in context.

![Diagram](image)

**Figure 1.2: Research questions for this research**

## 1.6 Research Methods and Studies

This research was conducted at several industrial partners within the GRIFFIN and Stephenson projects.

All industrial partners where we performed studies are involved in global software development. Furthermore, the organizations were interested in obtaining research results on (how to) better utilize architectural knowledge in their GSD activities.

Although the GSD involvement and the interests of the industrial partners provides for the ability to use results obtained in a certain case study in subsequent studies, we do acknowledge that differences exist between the industrial partners in e.g., industry domain and size of the development organizations. To overcome these differences, we explicitly decided to not make an *a priori* assumption on the development process or
any methodology used by the industrial partners or any other organization that wants to implement architectural knowledge management in GSD.

In this section, we provide an overview of the research methods we applied in this research. To this end, Table 1.1 provides an overview of the studies performed, the research methods applied, the research questions addressed, and the chapter in this thesis that describes the results. The research methods applied are described in §1.6.1. Next, the outline and high-level approach of each of the individual studies is described in §1.6.2 through §1.6.7.

Table 1.1: Overview of studies and research methods

<table>
<thead>
<tr>
<th>Study</th>
<th>Research methods</th>
<th>Research questions</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. How is architectural knowledge shared?</td>
<td>Exploratory survey</td>
<td>RQ1</td>
<td>2</td>
</tr>
<tr>
<td>2. Understanding practices for AKM in GSD</td>
<td>Exploratory case study, Document analysis, Exploratory survey, Interviews</td>
<td>RQ2</td>
<td>3, 4</td>
</tr>
<tr>
<td>3. Identifying product practices for AKM in GSD</td>
<td>Document analysis, Critical analysis of the literature</td>
<td>RQ2</td>
<td>5</td>
</tr>
<tr>
<td>4. Identifying practices for AKM in GSD</td>
<td>Critical analysis of the literature, Descriptive case study, Interviews</td>
<td>RQ2</td>
<td>6, 7</td>
</tr>
<tr>
<td>5. (Semantic) wiki support for AKM in GSD</td>
<td>Critical analysis of the literature, Prototype development</td>
<td>RQ3</td>
<td>8</td>
</tr>
<tr>
<td>6. Understanding what and how architectural knowledge is managed</td>
<td>Descriptive case study, Interviews, Content analysis</td>
<td>RQ4</td>
<td>9</td>
</tr>
</tbody>
</table>

1.6.1 Research Methods

**Exploratory survey** – (Pfleeger and Kitchenham, 2001) define a survey as “a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes, and behavior”. Thus, a survey can be used to attempt to describe a phenomenon of interest (Kitchenham and Pfleeger, 2001-2002). An exploratory survey is a specific kind of survey aimed at performing an exploratory field study in which there is no test of relationships between variables (Holz et al., 2006). We used an exploratory survey in our first study to identify how architectural knowledge is used and in a descriptive case study on the perceived usefulness of the architectural knowledge management practices for GSD at Organization D, a large IT service provider.

**Case studies** – The main research method applied in for the research described this thesis is case study research. We use the definition of a case study provided by (Yin, 2003): “a case study is an empirical inquiry that investigates a contemporary phenomenon
within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident”. As such, a case study is a research strategy which focuses on understanding the dynamics present within single settings (Eisenhardt, 1989).

Case study research emphasizes a detailed contextual analysis of a limited number of events or conditions and their relationships. According to (Yin, 2003), case studies may help in contributing to our knowledge of individual, group, organizational, social, political, and related phenomena. Case studies are the preferred strategy when ‘how’ or ‘why’ questions are being posed and when the researcher has little control over events. Different types of case studies exist (Yin, 2003):

- **Explanatory case studies** – These studies are causal investigations and focus on the understanding of why a certain phenomenon occurs by studying potential factors that led to it.

- **Exploratory case studies** – Study patterns in collected data to identify a model that can be used to view or interpret the data.

- **Descriptive case studies** – Study aimed at validating an initial theory or model that may be the result of exploratory investigations.

We performed case studies where we conducted an empirical study at one of the industrial partners of our research; the case studies were primarily exploratory and descriptive in nature. Our aim was either to identify a model for interpretation or to test or validate the model developed initially. The case study performed at Organization A, a large developer of consumer electronics products, was exploratory in nature, the case studies performed at Organization D and Organization E (a Dutch producer of high-end printers for the business markets in high-volume printing, wide-format printing, and office printing) were descriptive in nature.

**Document analysis** – Document analysis was performed on the architectural rules in the case study at Organization A and on the set of software product assessment reports to identify differences between GSD projects and collocated projects.

**Critical analysis of the literature** – This method concerns an appraisal of relevant published material based on careful analytical evaluation. We performed a critical analysis of the literature to identify architectural knowledge management practices based on the requirements engineering literature. We used a systematic literature review (Brereton et al., 2007) for our study to identify the key elements for architectural knowledge as part of our research on product practices for architectural knowledge management in GSD and on wiki support for the architectural knowledge management practices in GSD.

**Prototype development** – Prototype development is a technique that is used to validate parts of the theories developed for their suitable application in practice. The system that is built does not comprise all envisioned features; yet, it provides sufficient functionality to convince the reader that the application is viable and can be effective. We have developed a prototype to show how selected use cases for architectural knowledge on which several of the architectural knowledge management practices for GSD build, can be implemented in practice.
1.6. Research Methods and Studies

**Content analysis** – Content analysis is a research method that uses a set of procedures to make valid inferences from text (Weber, 1990). Content analysis enables researchers to analyze large amounts of textual information and systematically identify its properties, e.g., the frequencies of most used keywords (conceptual analysis, according to Weber, 1990). This research method helps in e.g., revealing the focus of individuals or groups. We used this research method at Organization E to identify what architectural knowledge management practices for GSD are actually used in practice.

In several case studies, we use interviews as an information gathering technique. In an interview, questions are posed by an interviewer to an individual or group of individuals. In our research, we have performed individual interviews using a question set. Conducting individual interviews allows for unbiased results and prevents group think. Group think is “a mode of thinking that people engage in when they are deeply involved in a cohesive in-group, when the members’ striving for unanimity overrides their motivation to realistically appraise alternative courses of action” (Janis, 1972). A question set ensures that the same general areas of information are collected from each interviewee and hence allows us to compare and combine the results. We conducted interviews as part of all the case studies performed in this research; at Organization A, Organization B (a Dutch IT service provider that develops business information systems), Organization D, and Organization E.

1.6.2 1 – Survey: How is architectural knowledge shared?

Our first study was conducted in 2006 and builds upon earlier work performed in the GRIFFIN project. As part of the initial activities in the GRIFFIN project, we have developed a series of so-called use cases for architectural knowledge (van der Ven et al., 2006a). These use cases cover a variety of reasons, activities, and motivations (i.e., possible uses) for using architectural knowledge.

The intent of this study was to obtain an understanding of the applicability of these use cases in the architecting community in The Netherlands. We performed survey-based research at a broad selection of IT practitioners with architecting responsibilities in the Netherlands.

One of the main insights obtained in this study is that the intended benefits of viewing the architecture as a set of architectural decisions (see 1.1.1) is not yet embedded within the mindset of the architects. Typically, the mindset of architects is rather positive and aimed at forward engineering by taking architectural decisions to address stakeholders’ concerns, rather than (re-)viewing the set of taken architectural decisions to perform impact or risk analyses.

---

2Hence, the term use case differs from the definition given in Jacobson (1992) which focuses on a series of actions. Although the use cases for architectural knowledge are described as a series of steps, this first study primarily focuses on the possible uses.
1. Introduction

1.6.3 2 – Case study: Understanding practices for architectural knowledge management in global software development

The first case study at one of GRIFFIN’s industrial partners (Organization A) was conducted in 2006. The aim of this case study was to understand how architectural knowledge is managed in a typical organization involved in global software development. In this case study, we specifically focused on architectural knowledge that must be complied with at all times across all development sites. This architectural knowledge serves as architectural rules to the respective development sites.

We have researched the role of the architectural rules and the compliance practices that are implemented at a department involved in GSD at Organization A. During this research, we worked closely with the central architecture team. Our activities consisted of analyzing the use of the architectural rules and the role of the architecture team in securing compliance with these rules. We conducted questionnaires and follow-up interviews as data collection techniques and developed recommendations for improved compliance.

This case study exemplified that, apart from a good structure of the architectural knowledge itself, additional process practices are needed to let the architectural knowledge sink in properly and ensure compliance with the architectural knowledge across sites: these practices included frequent communication on the architectural knowledge, documentation of possible deviations, and verification of compliance with the architectural knowledge.

To further substantiate the results obtained in the study at Organization A, we compared Organization A with a second organization, Organization B, in 2007. We constructed a comparison model based on the challenges and issues involved in GSD. We used this model to compare how Organization A and Organization B each used architectural knowledge management practices to overcome these challenges and issues.

The results showed that whereas Organization A has a focus towards using rules related to compliance, Organization B more reverts to measures and practices in the process to overcome the challenges and issues involved in GSD.

1.6.4 3 – Study: Identifying product practices for managing architectural knowledge in global software development

The research as described in §1.6.3 led us to believe that the industrial partners where we performed these studies have a focus on practices that relate to the development process exists. An alternative approach towards managing architectural knowledge is capturing architectural knowledge in the product. This third study, performed in 2008-2009, was aimed at identifying possible product practices for architectural knowledge management in global software development.

We used the results of a series of software product assessments performed by an industrial partner in the GRIFFIN research, Organization C. This organization has laid down its experience in performing over 20 of these assessments in a reusable framework containing evaluation criteria (i.e., a software quality evaluation framework). These evaluation criteria are backed by literature as to positively impact quality attributes (as
1.6. Research Methods and Studies

e.g., specified in [ISO/IEC, 2001]) and hence serve as measures that can be taken in the software product to increase its quality.

Some of the products included in our study were developed using GSD, others were developed using collocated software development. We compared each of these two groups using a reference framework. This reference framework consisted of the set of evaluation criteria used in the assessments that concern architectural knowledge and of key elements of architectural knowledge that were distilled via a critical analysis of the literature.

Our main discovery during this research was an observed indifference between the use of architectural knowledge in the product in GSD and collocated software development. Several observations were made on the application of architectural knowledge in products developed using GSD.

1.6.5 4 – Case study: Identifying practices for architectural knowledge management in global software development

This case study, performed in 2008, was aimed at defining practices for the management of architectural knowledge in global software development. Our previous studies have provided insight into practices that are typically performed at the organizations involved in these studies. This study was aimed at identifying and proposing a more generic collection of practices for architectural knowledge management in GSD.

We prepared the case study by performing a critical analysis of the requirements engineering literature on solutions to overcome challenges in GSD. We collected a combined series of requirements engineering practices and translated these to the disciplines of architecture and architectural knowledge management. This research resulted in a collection of seven practices. Most of these practices support a personalization strategy towards knowledge management, a minority supports a codification strategy.

The combined set of practices identified through the literature analysis was used as input for the case study. The case study’s aim was to perform a validation of the perceived usefulness of these practices at GSD projects of an industrial partner in the GRIFFIN project, Organization D. In addition, we related the perceived usefulness of these practices to the number of sites in the GSD projects. Before sending out a survey to all Dutch architects employed at Organization D, we conducted several semi-structured interviews to collect an initial view of the software engineering practices that were in place at Organization D.

The study showed that the architectural knowledge management practices in general are perceived as useful. The personalization practices are perceived as more useful than the codification practices. Moreover, this study showed a peak in perceived usefulness at projects that run with three sites. We found that these several of these software development projects were not started with three sites but evolved into that stage after having run for some time with two sites. The results denote a need to implement architectural knowledge management practices for global software development proactively to prevent problems with architectural knowledge sharing from arising.
1.6.6 5 – Study: (Semantic) wiki support for architectural knowledge management in global software development

Architectural knowledge management can be supported using a variety of tool-supported mechanisms to ease its adoption and use. In this study, performed in 2010, we first identified to what extent wikis can be used to support architectural knowledge management in global software development. Wikis have proven to provide support in knowledge sharing and community building. As such, employing wikis may help to overcome several challenges (e.g., those related to geographical distance) involved in GSD.

In this study, we found that wikis form a good mechanism to implement a hybrid strategy for managing architectural knowledge (focused on both codification and personalization knowledge management practices) in GSD. Furthermore, we discovered that a substantial part of the AKM practices may actually be implemented using wikis.

To further substantiate the results obtained in this study, a prototype implementation of an architectural knowledge sharing environment was developed in 2010. Instead of using a wiki for this prototype, we have used a semantic wiki for the prototype implementation. Semantic wikis combine wiki properties such as ease of use, open collaboration, and linking with Semantic Web technologies such as structured content, knowledge models in the form of ontologies, and reasoning support based on formal ontologies with reasoning rules (Schaffert et al., 2008; Liang et al., 2009). For the prototype, we developed a software engineering ontology and implemented three use cases (i.e., possible uses) for architectural knowledge using this ontology. The use case of software reuse supports the architect to if existing software can be reused to implement a new functional requirement. The use case changing requirement supports the architect in updating an architecture design according to a changing requirement. The use case design impact evaluation supports the architect in evaluating and identifying the impact of changing requirements on architecture design.

This study has brought us the insight that wikis, and semantic wikis in particular, are powerful means to capture architectural knowledge and reason with (query, select) the combined set of knowledge to address stakeholders’ concerns or perform architectural design activities in GSD.

1.6.7 6 – Case study: Understanding how architectural knowledge is managed

In the previous studies, we have collected a set of practices for the management of architectural knowledge in global software development. The results of these studies have strengthened us to believe that the identified practices are suitable for use in GSD.

In this descriptive case study, conducted at Organization E as part of the Stephenson project in 2010-2011, we intended to validate this theory: the actual use of the collected set of practices for architectural knowledge management for global software development that are the result of research described in this thesis.

We used a questionnaire with the aim to explore and understand the way communication, coordination, and knowledge sharing occurs at the activities performed at Or-
The questionnaire was used as the basis for conducting interviews at the three major development sites at Organization E.

The results show that Organization E emphasizes architectural knowledge management practices that promote decentralization and, as a consequence, personalization (as opposed to centralization viz. codification). We identified one new useful practice: ‘peered sites’, covering a combination of activities that support a balance in decision-making power across sites.

1.7 Publications

The research presented in this thesis has been published previously at conferences and workshops. The publications are included in subsequent chapters as-is, with the exception of added chapter introductions (in italics) and some minor corrections of future work paragraphs. The research reported in this thesis has been performed by Viktor Clerc as the prime researcher (except for the papers that constitute Chapter 8; this research was performed in collaboration with Edwin de Vries, Antony Tang, and Peng Liang) and have been written down by Viktor. Most of the publications are co-authored by Patricia Lago and Hans van Vliet. For these publications, Patricia and Hans provided frequent suggestions on the publication structure and various other review comments.

Parts of Chapter 2 have been previously published as:


Parts of Chapter 3 have been previously published as:


Parts of Chapter 4 have been previously published as:


Parts of Chapter 5 have been previously published as:

1. Introduction

Parts of Chapter 6 have been previously published as:


Parts of Chapter 7 have been previously published as:


Chapter 8 consists of two previously published papers. The paper with Edwin de Vries and Patricia Lago is based on research performed by Edwin. Viktor extended his work in providing the comparison between wikis and functionalities. The book chapter with Antony Tang, Peng Liang, and Hans van Vliet has been co-authored by Antony, Peng, and Viktor and reviewed by Hans. The parts of the book chapter included in this thesis were written by Viktor.


Parts of Chapter 9 have been previously published as:

The Mindset of Architects

In this chapter, we describe how practitioners in the Netherlands view and use architectural knowledge. We have investigated how several proposed use cases for architectural knowledge are actually used by practitioners with architecting responsibilities in Dutch various organizations. This study reveals the mindset of practitioners with respect to the use of architectural knowledge: we list what use cases are important for what architecting roles and on what architecture levels.

2.1 Introduction

A software architecture is a transferable abstraction of a system and allows for communication of that system to different stakeholders (Bass et al., 2003). Software architecture and software architecture practices are gaining importance since they enable reasoning on the design of the system and verifying quality attributes of a system at an early stage in the development cycle.

Rather than viewing the software architecture as a set of components and connectors, recent literature regards the software architecture as the set of architectural design decisions (Bass et al., 2003; Jansen and Bosch, 2005). The collection of architectural design decisions and the resulting design together constitute architectural knowledge (Kruchten et al., 2006). Besides providing insight into the current software architecture, architectural knowledge also caters for the ‘why’ of the software architecture, its rationale.

To get the most out of the architectural knowledge of information systems in general, we need to determine how different stakeholders use architectural knowledge. We term these typical uses use cases for architectural knowledge. Some of these use cases may depend on the roles that stakeholders fulfill or the architecture level stakeholders are engaged in. Architects may favour other use cases than designers or technical specialists, and enterprise architecture practitioners may give priority to other use cases than software architecture practitioners. Currently, we determine what information is particularly important for certain stakeholders, by using approaches and standards such

---

1As indicated in §1.6.2, this definition differs from the definition for the use case technique as part of OMG’s Unified Modeling Language standard.
2. The Mindset of Architects

as (Clements et al., 2003; IEEE, 2000). Yet, we do lack insight into how these stakeholders use architectural knowledge.

We conducted a survey-based study to address the lack of insight into the importance of architectural knowledge. We designed a survey which includes use cases for architectural knowledge. These use cases are based on earlier work (Kruchten et al., 2006; van der Ven et al., 2006a), experiences in industry, and our own experience.

This chapter reports on the results of our study. We provide insight into the way practitioners in the Netherlands view and use architectural knowledge. In doing this, we reveal the mindset of practitioners with respect to the use of architectural knowledge by listing what uses are important for what roles and on what architecture levels.

Based on the survey results, we make the following observations. Architects regard the architecting process as a forward engineering discipline and do not see clear benefits of reflection, assessment, and change of the architecture. Yet, literature argues that these are precisely the intended benefits of architecture (Bass et al., 2003). Apparently, these intended benefits of architecture have not yet been firmly established in the mindset of architects nor transferred to practice. Furthermore, a forward decision-making process is reflected by the mindset of architects, but the value of managing the set of decisions (so, architectural knowledge management) is not yet clear. Finally, the importance of stakeholder communication of the architecture is generally recognized.

The results of this research call for further knowledge transfer on the more innovative concept of viewing architecture as a set of architectural decisions. Furthermore, it is important to quantify the benefits of this concept. At the same time, further research is needed into the foundation for the mindset to identify the activities needed to further establish the concept of architectural knowledge in the architect’s mindset.

The remainder of this chapter is organized as follows. In §2.2 we discuss related work in the field of architectural knowledge, design rationale, and architectural roles. Next, §2.3 describes the design of the research. §2.4 and §2.5 present the findings and a discussion of their limitations. In §2.6 we reflect on the results and provide conclusions. Finally, §2.7 provides directions for future work.

2.2 Related Work

Tang et al. (2006), have analyzed the use and documentation of architecture design rationale. The survey reveals the view of the participants on several generic uses of design rationale. The study shows that although participants regard design rationale as important, they do not capture the rationale. The main reason for this is a lack of appropriate tools to support the architects. Furthermore, the study shows that architects tend to focus on the positive aspects of architectural decisions and design rationale instead of looking for problems in a specific architecture. We view design rationale as a specific subset of architectural knowledge (Kruchten et al., 2006). In this chapter, we revert to the use cases for architectural knowledge, initially described in (Kruchten et al., 2006) and elaborated on in (van der Ven et al., 2006a), and provide a detailed view on possible uses of architectural knowledge.

A template for architectural decisions is provided by (Tyree and Akerman, 2005).
2.3. Research Design

This template describes a decision, its underlying assumptions, related requirements, and implications. The template is useful for organizing architectural knowledge but does not provide insight into the use of architectural knowledge. Multiple templates or tailoring of existing templates may be necessary to fully support architects in their use of architectural knowledge. Our work can provide input for this. Zimmermann et al. report on a framework that can be useful in identifying, making, and enforcing architectural decisions during the architecting process (Zimmermann et al., 2007).

Smolander (2002) extensively describes the meaning of architecture in practice. Smolander describes four metaphors for architecture: ‘architecture as blueprint’, ‘architecture as literature’, ‘architecture as language’, and ‘architecture as decision’. The metaphors explain the meaning of architecture in practice. This meaning may differ depending on the role practitioners fulfill or the architectural levels they are engaged in. We provide insight into the importance of use cases for architectural knowledge and relate this insight to the metaphors from (Smolander, 2002). Thus, we show what metaphors are accepted and in use by practitioners.

Hofmeister et al. (2000) provides a good description of possible roles and activities of an architect. Examples include a visionary, technical consultant, decision maker, and coach. These different activities could be supported by appropriate use cases for architectural knowledge. Use cases for architectural knowledge that are deemed important for aforementioned roles enable the architect to effectively capture and reason about architectural knowledge. The other way around, the relevance of use cases for architectural knowledge for the practitioners could indicate to what extent the possible roles and activities from (Hofmeister et al., 2000) in fact are established.

Clements et al. (2007) performed a study on publicly available resources to identify the duties, skills, and knowledge of software architects. They show that the role of an architect implies more than only making technical decisions. Rather than focusing on the competences of architects, our study focuses on the use of architectural knowledge to support the architect in achieving and maintaining his competences.

2.3 Research Design

We aim to find out how practitioners engaged in architecture in the Netherlands view architectural knowledge. This helps us to construct the mindset of architects with respect to architectural knowledge. In order to reveal this mindset, we use a survey instrument. Survey instruments are widely used in software engineering research (Kitchenham and Pfleeger, 2001-2002). In survey-based research, a number of factors should be taken into account: the design of the survey, the selection of participants, and how to control the response rate of the survey. The remainder of this section describes how we have addressed these factors.

2.3.1 Survey design

The ultimate objective of the survey is to identify the way practitioners view and use architectural knowledge. We took the typical uses distilled from experienced practitioners
within the GRIFFIN research project (Kruchten et al., 2006; van der Ven et al., 2006) as a starting point. We validated and augmented the list of use cases with the industrial partners in our research project and with our own experience. Next, we reformulated each use case into a clear, self-explaining statement on architectural knowledge. We allowed the survey participants to indicate the importance of each use case using a 5-point Likert scale (Likert, 1932) ranging from ‘not important’ to ‘very important’.

We hypothesized that the importance of certain use cases may depend on the roles practitioners fulfill and on the level of architecture that practitioners are engaged in. We posed some demographic questions and specifically asked the participants to a) indicate the architectural roles they typically fulfill and b) indicate the relative amount of time they spend on certain architecture levels. Using the information on the roles and architecture levels, we constructed two different perspectives to analyze the way respondents perceive architectural knowledge.

As a first step, we conducted a pilot with the survey on a focused group consisting of selected employees of one of our industrial partners. We then developed a web-based survey for administration of the complete population.

2.3.2 Selection of participants of the survey

We needed to construct a representative subset of Dutch practitioners that play a key role in architecturing activities. To come to this subset, we identified three dimensions by which we selected participating organizations: domain (such as banking, telecommunications, insurance, governmental), type (IT service providers, software development organizations), and market (commercial, non-commercial). Next, we selected organizations or platforms (such as communities of enterprise architects and embedded architects) in each of these dimensions and identified practitioners that play a key role in architecture at these organizations. This gives us confidence that we have selected a representative subset of Dutch practitioners to give us feedback on the use of architectural knowledge.

2.3.3 Control of the survey

In order to keep control on the response rate, we directly contacted key practitioners at the organizations involved. We enquired their willingness to act as on-site representative for that organization. We sent these representatives the hyperlink to the online survey. The on-site representatives forwarded the hyperlink to knowledgeable colleagues and notified us of the total number of colleagues involved (snowball sampling (Kitchenham and Pfleeger, 2001-2002)).

2.4 Findings

This section describes our findings. We first provide demographic information in §2.4.1 after which we discuss the two different perspectives for presenting the results as described in §2.3.1. Next, §2.4.4 describes how we clustered the use cases for architectural
2.4. Findings

knowledge. After that, we present our main results in §2.4.5 how practitioners view and use architectural knowledge.

2.4.1 Demographic information

We sent out the survey to 36 persons acting as respondents and on-site representatives. They forwarded the survey to 348 practitioners. So, in total, 384 practitioners were reached. We collected 143 responses, of which 107 were complete. This corresponds to a response rate of 27.86%. We took these as a basis for our survey.

Of the total population, 213 practitioners are employed at one of the large IT service providers included in our survey. We discuss the overrepresentation of practitioners employed at IT service providers in §2.5. We received 75 responses from these practitioners, corresponding to a response rate of 35.21%. The remaining 171 practitioners are employed at smaller IT consultancy firms or e.g., IT departments of banks, insurance organizations or governmental organizations. We received 32 responses, corresponding to a response rate of 18.71%.

2.4.2 Architecture levels

Various different definitions of ‘architecture’ exist (SEI, 2011). Examples include a systems-oriented view and a view focusing on the information flow in or surrounding a software system. In our survey we used concise definitions from (Florijn et al., 2003) for the so-called architecture levels:

- **Software architecture** – The structure and relations of a software system.
- **Systems architecture** – The architecture of a single system, taking into account both software and hardware.
- **Information architecture** – The information needs and flows of business functions as they are identified.
- **Enterprise architecture** – Architecture at the level of an organizational unit or an organization as a whole.
- **Process architecture** – A description of the processes running in or surrounding a software system.

Each practitioner indicated the amount of time spent on a certain level of architecture. To be able to compare the data collected, we normalized the total amount of time spent by each practitioner to 100%. The relative amount of time spent on each level of architecture for all respondents is depicted in Table 2.1. We observed that a concrete architecture of a single system (i.e., ‘software architecture’ and ‘systems architecture’) receives more attention than company-wide architectures (i.e., ‘information architecture’, ‘enterprise architecture’, and ‘process architecture’). Other architecture levels that are less often practiced are grouped into the ‘other’ category in Table 2.1.

Since the population is relatively large, we grouped the respondents into clusters and based our analysis on the clusters instead of on the individual responses. Of course,
2. The Mindset of Architects

Table 2.1: The percentage of time spent by all respondents on architecture levels

<table>
<thead>
<tr>
<th>Level of architecture</th>
<th>Percentage of time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software architecture</td>
<td>26.91%</td>
</tr>
<tr>
<td>Systems architecture</td>
<td>23.98%</td>
</tr>
<tr>
<td>Information architecture</td>
<td>15.80%</td>
</tr>
<tr>
<td>Enterprise architecture</td>
<td>12.36%</td>
</tr>
<tr>
<td>Process architecture</td>
<td>11.06%</td>
</tr>
<tr>
<td>Other</td>
<td>9.88%</td>
</tr>
</tbody>
</table>

Architects may work on different architecture levels simultaneously. We wanted to see if these architecture levels are related based on the responses of the practitioners. For example, if architects often work on two architecture levels simultaneously, we group these two levels. Moreover, this enables us to group architecture levels that have different names, but in fact appear to be closely related. In order to group similar or closely related architecture levels, we calculated the correlation between each of the architecture levels. This resulted in an n-dimensional space, n being the number of architecture levels. In order to plot the relative distances between the architecture levels, we reduced the number of dimensions to two using classic multi-dimensional scaling [Borg and Groenen, 2005]. Multidimensional scaling refers to representing the architecture levels as points, in such a way that the interpoint distances approximate the specified dissimilarities. In order to assess the accuracy of the distance, we applied k-means clustering [MacQueen, 1967] to cluster architecture levels with a small distance. For data that lie in Euclidean space, k-means clustering tries to find a partition that minimizes the sums of squared errors about the cluster means, which represent their respective clusters. We observed that the optimal distribution of architecture levels in clusters occurred when we used five clusters. We compared the clusters that appeared with k-means clustering with the distance plot provided by classic multi-dimensional scaling. The comparison revealed that we selected the right number of clusters and that we found the correct distribution of elements over the clusters. Consequently, the clusters contain elements that are different in nature and have their overlap reduced to a minimum. We observed the elements in these clusters of architectural levels and labeled the clusters.

The results are shown in Table 2.2. The table shows that distinct clusters provide for the relation of a software system and the hardware it runs on (Systems architecture), the structure of a software system (Software architecture), the structure of the organization or department using the software system (Enterprise architecture), and the process and information flow in or surrounding a software system (Information and process architecture).

Of the most significant architecture levels, only Information architecture and Process architecture are very often worked on by a single respondent simultaneously. Consequently, they fall into the same cluster. The remainder of the most significant architecture levels each fall into a distinct cluster.
2.4. Findings

Table 2.2: Clusters of architecture levels

<table>
<thead>
<tr>
<th>Cluster label</th>
<th>Levels of architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems architecture</td>
<td>systems architecture</td>
</tr>
<tr>
<td>Software architecture</td>
<td>software architecture, management of architecture</td>
</tr>
<tr>
<td>Enterprise architecture</td>
<td>enterprise architecture</td>
</tr>
<tr>
<td>Information and process architecture</td>
<td>information architecture, process architecture</td>
</tr>
<tr>
<td>Other</td>
<td>development coach, integration architecture, infrastructure architecture, service architecture, maintenance of architecture, solution architecture</td>
</tr>
</tbody>
</table>

Practitioners can potentially work on different levels of architecture simultaneously. In spite of that, according to the clusters shown in Table 2.2, practitioners do not do this. They are specialized in working at one specific level of architecture only. Possible reasons for this are the different technical and interpersonal skill-sets required at each architecture level. For example, practitioners who mainly work on the level of Systems architecture are concerned with CPU performance, interrupt levels, and other technical topics, whereas practitioners who mainly work on the level of Process architecture are concerned with implications of decisions on working processes, which places less requirements on technical skills. Required interpersonal skills can vary at different architecture levels as well. As the topics that require to be communicated get less technical, the potential audience could grow. Consequently, the set of stakeholders with which to communicate grows from technology-oriented stakeholders to include more business-oriented stakeholders as well.

2.4.3 Architectural roles

The participants indicated the architectural roles they typically fulfill. The survey contained a list of roles, including ‘architect’, ‘reviewer of architecture’, ‘project manager’, and ‘developer’.

We repeated the same analysis as described in §2.4.2 to identify clusters of roles typically fulfilled by a single respondent. The optimal distribution of architectural roles in clusters occurred when we used five clusters. Again, we labeled the clusters. The results are listed in Table 2.3.

The clusters labeled High-level and Low-level show that, apparently, architectural roles are related based on level of abstraction with respect to a software system and practitioners work at one specific level of abstraction. Our results also show that practitioners generally do not switch between the different levels. This contradicts with the view on the role of a software architect as an implementor (Hofmeister et al., 2000); according to our survey, architect do not design or implement that often. Furthermore, the roles ‘architectural educator’ and ‘project manager’ share a communication responsibility towards a variety of stakeholders. Consequently, we label this cluster Commu-
2. The Mindset of Architects

Table 2.3: Clusters of architectural roles

<table>
<thead>
<tr>
<th>Cluster label</th>
<th>Architectural roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicator</td>
<td>architectural educator, project manager</td>
</tr>
<tr>
<td>Low-level</td>
<td>designer, developer, reviewer of source code</td>
</tr>
<tr>
<td>Specialist</td>
<td>consultant, technical specialist</td>
</tr>
<tr>
<td>High-level</td>
<td>architect, reviewer of architecture</td>
</tr>
<tr>
<td>Other</td>
<td>end user, lead architect, security consultant, other</td>
</tr>
</tbody>
</table>

2.4.4 Clustering the use cases

We listed the use cases for architectural knowledge from (van der Ven et al., 2006a) and asked the practitioners to indicate the importance of each use case for their daily work and whether they actually performed the use case. We used the answers of participants of the use cases to reveal an underlying structure in the use cases. The structure would excavate similarities between use cases based on the answers and would allow us to cluster the use cases accordingly.

First, we used principal components analysis (Anton, 2005) to identify the underlying structure in the use cases for architectural knowledge based on the respondents’ answers. We could not find any underlying structure; the variance in the scores of the use cases was explained by one main principal component.

Since the principal components analysis did not lead to a clustering of the use cases, we next tried to cluster the use cases based on the purpose of the individual use cases. Most use cases for architectural knowledge could be clustered relatively easily, e.g., some use cases clearly dealt with stakeholders only. Consequently, we grouped these use cases into a single cluster. For some use cases, clustering was more difficult. These use cases could be grouped into multiple clusters, e.g., ‘add an architectural decision’ could point at a forward architecting approach, but at the same time assumes that a set of architectural decisions exists to which the new decision is added as well – see Table 2.4. We identified the most appropriate cluster for these use cases by analyzing the questionnaire results of the participants for these use cases. We compared the answers on a use case with the average of the answers for each candidate cluster. We assigned the use case to the cluster with the highest similarity in answers (see §2.4.5). The interpretation of the survey results also led to the cluster labels. Table 2.4 lists the resulting clusters of use cases for architectural knowledge.

The use case cluster Architectural decision set presupposes that a set of knowledge entities (i.e., architectural decisions) and relations between these knowledge entities exist (see (Kruchten et al., 2006)) for a list of possible relations). The use cases in this cluster are aimed at managing that set. Several other use cases have to do with assessing or reviewing an architecture. Within this Assessment cluster, we distinguish between use cases that imply a forward-engineering approach to architecture (i.e., from
### 2.4. Findings

<table>
<thead>
<tr>
<th>Use case cluster</th>
<th>Use cases</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architectural decision set</strong></td>
<td>View the change of the architectural decisions over time</td>
</tr>
<tr>
<td></td>
<td>Recover architectural decisions</td>
</tr>
<tr>
<td></td>
<td>Identify incompleteness</td>
</tr>
<tr>
<td></td>
<td>Detect patterns of architectural decision dependencies</td>
</tr>
<tr>
<td></td>
<td>Check for superfluous architectural decisions</td>
</tr>
<tr>
<td></td>
<td>Cleanup the architecture</td>
</tr>
<tr>
<td><strong>Assessment – reqs.→arch.→impl.</strong></td>
<td>Check implementation against architectural decisions</td>
</tr>
<tr>
<td></td>
<td>Check correctness (i.e., architecture versus requirements)</td>
</tr>
<tr>
<td></td>
<td>Evaluate the impact of an architectural decision</td>
</tr>
<tr>
<td></td>
<td>Evaluate consistency</td>
</tr>
<tr>
<td></td>
<td>Get consequences of an architectural decision</td>
</tr>
<tr>
<td><strong>Assessment – risk, trade-off analysis</strong></td>
<td>Perform a review for a specific concern</td>
</tr>
<tr>
<td></td>
<td>Perform an incremental architectural review</td>
</tr>
<tr>
<td></td>
<td>Assess design maturity</td>
</tr>
<tr>
<td></td>
<td>Conduct a risk analysis</td>
</tr>
<tr>
<td></td>
<td>Conduct a trade-off analysis</td>
</tr>
<tr>
<td><strong>Stakeholder-centric</strong></td>
<td>Identify the subversive stakeholder</td>
</tr>
<tr>
<td></td>
<td>Identify key arch. decisions for a specific stakeholder</td>
</tr>
<tr>
<td></td>
<td>Identify affected stakeholders on change</td>
</tr>
<tr>
<td></td>
<td>Identify unresolved concerns for a specific stakeholder</td>
</tr>
<tr>
<td></td>
<td>Keep up-to-date</td>
</tr>
<tr>
<td></td>
<td>Inform affected stakeholders</td>
</tr>
<tr>
<td></td>
<td>Identify important architectural drivers</td>
</tr>
<tr>
<td><strong>Forward Architecting</strong></td>
<td>Retrieve an architectural decision</td>
</tr>
<tr>
<td></td>
<td>Add an architectural decision</td>
</tr>
<tr>
<td></td>
<td>Remove consequences of a cancelled decision</td>
</tr>
<tr>
<td></td>
<td>Reuse architectural decisions</td>
</tr>
</tbody>
</table>

requirements, to architecture, to implementation), and use cases that target at performing different kinds of analyses and reviews. The first set aims at verification of the architecting activities (“are we still on the right track?”) whereas the second set aims at validation. Seven use cases form the cluster **Stakeholder-centric**. These use cases concern identification of stakeholders and communication of the architecture to specific stakeholders. The cluster **Forward architecting**, finally, consists of use cases that create, request, reuse or remove architectural decisions.
2. The Mindset of Architects

2.4.5 Participants’ views on the use cases

Instead of elaborating on each of the 107 responses individually, we took the clusters of architecture levels and architectural roles as described in §2.4.2 and §2.4.3 as two perspectives for analyzing the survey results.

We built a structure to be used to identify the importance of a specific use case for architectural knowledge to a specific cluster (of architectural roles or architecture levels) as follows. For each respondent \(i\) in a cluster with \(n\) respondents, we used the Likert scores (\(score_i\)). Using the relative contribution of the respondent to that cluster (\(\%_i\)), we calculated the weighted average as shown in (2.1):

\[
score = \frac{\sum_{i=1}^{n} score_i \cdot \%_i}{\sum_{i=1}^{n} \%_i}
\] (2.1)

Next, we identified outliers by defining an upper and lower limit of importance: within the possible range of scores from 1 – 5 we regard a use case with a score of \(\geq 3.5\) as ‘important’ and a use case with a score of \(\leq 2.5\) as ‘not important’. The results are listed in Table 2.5. Each row in Table 2.5 relates a cluster of use cases for architectural knowledge to both the clusters of architectural roles and the clusters of architecture levels. The importance of each use case cluster for each cluster of architectural roles and each cluster of architecture levels is provided. Important clusters are marked ‘(+)’, not important clusters are marked ‘(−)’. Impartial results are not listed in the table. The findings are discussed below. An extensive discussion of their implications is given in §2.6.

Architectural decision set – The use cases for architectural knowledge within the cluster Architectural decision set assume that a set of architectural decisions is at the practitioner’s disposal. In terms of the use cases, architecting thus boils down to managing and manipulating that set of architectural decisions. Table 2.5 shows that viewing architectural knowledge as a set of decisions has not been established at the Software architecture and Systems architecture levels. Furthermore, viewing the architecture as a set of decisions is regarded as not important for Communicator and Specialist roles. High-level and Low-level roles (i.e., ‘architects’ versus ‘designers’ and ‘developers’) deem these use cases neutral. Apparently, the view on architecture as a set of architectural decisions (Jansen and Bosch, 2005; Jansen, 2008) and managing that set has not yet transferred to practice, nor is it of particular value to the practitioners.

Assessment – reqs.→arch.→impl. and Assessment – risk, trade-off analysis – The cluster labeled Assessment – reqs.→arch.→impl. covers traceability of architectural decisions to the actual implementation, the relation between decisions themselves, and from architectural decisions back to the requirements that have been set for the information system. Especially respondents who strongly contribute to the role clusters

\(^2\) 1 being not important, 5 being very important.
2.4. Findings

Table 2.5: Importance of use case clusters per cluster of architectural roles and cluster of architecture levels. (+) denotes importance, (–) denotes unimportance

<table>
<thead>
<tr>
<th>Use case cluster</th>
<th>Cluster of arch. roles</th>
<th>Cluster of architecture levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural decision set</td>
<td>(–) Communicator</td>
<td>(–) Software architecture</td>
</tr>
<tr>
<td></td>
<td>(–) Specialist</td>
<td>(–) Systems architecture</td>
</tr>
<tr>
<td>Assessment – reqs.→arch.→impl.</td>
<td>(+) High-level</td>
<td>(+) All levels</td>
</tr>
<tr>
<td></td>
<td>(+) Low-level</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(+) Specialist</td>
<td></td>
</tr>
<tr>
<td>Assessment – risk, trade-off analysis</td>
<td>(–) Specialist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(–) Communicator</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(–) Low-level</td>
<td>(-) Software architecture</td>
</tr>
<tr>
<td>Stakeholder-centric</td>
<td>(+) High-level</td>
<td>(+) Enterprise architecture</td>
</tr>
<tr>
<td></td>
<td>(–) Communicator</td>
<td>(+) Process and information architecture</td>
</tr>
<tr>
<td></td>
<td>(–) Low-level</td>
<td></td>
</tr>
<tr>
<td>Forward Architecting</td>
<td>(+) High-level</td>
<td>(+) All levels</td>
</tr>
<tr>
<td></td>
<td>(–) Low-level</td>
<td></td>
</tr>
</tbody>
</table>

High-level, Low-level and Specialist (see Table 2.3) regard these use cases as important. These roles are the ‘construction’ roles with respect to architecture. This confirms our idea that practitioners involved in the construction of architectures have a need for traceability of architecture. The use cases in the cluster Assessment – risk, trade-off analysis are not regarded as important by the High-level cluster of architectural roles. Furthermore, especially practitioners engaged in Software architecture regard the use cases in this cluster as not important.

A difference that exists between the two subclusters within Assessment could lie in the architect’s mindset. The results of the cluster Assessment – reqs.→arch.→impl. reveal a mindset with a linear (i.e., non-iterative) approach to designing an architecture that satisfies the posed requirements and subsequently have the implementation satisfy the architecture. Use cases that offer traceability in this approach are regarded as important. The use cases in the cluster Assessment – risk, trade-off analysis, on the other hand, all are aimed at having an intermediate period of reflection to verify what risks apply, or what quality attributes could be affected by certain architectural decisions. These use cases are not directly related to either requirements or implementation.

In summary, in contrast to e.g., [Bass et al., 2003; Hofmeister et al., 2000], who state that architecture offers a good means to assess the correctness and suitability of the desired solution, our results reveal architects regard the use cases for architectural knowledge in the Assessment – risk, trade-off analysis cluster as not particularly important. Literature points out that an architecture enables us to assess the design maturity, perform incremental, iterative design reviews, and periodically identify the largest risks pertaining to the architecture. Apparently, these benefits of architecture are not valued
2. The Mindset of Architects

by our respondents, which is surprising.

Moreover, the use cases in the cluster Assessment – risk, trade-off analysis aim at finding possible problems in a certain architecture. Since practitioners do not regard these use cases as important, we might infer that practitioners do not favour a period of reflection in which the current state of the architecture is explicitly tested. Yet, this is one of the main reasons stated in the literature for developing an architecture (Bass et al., 2003). Apparently, these intended benefits of architecture have not yet been firmly established in the mindset of architects. The lack of value contributed to the intended benefits reveals a mindset of positiveness (“architects always take the right decisions”), which supports the findings of (Tang et al., 2006). Respondents do not like to use architectural knowledge to identify potential weaknesses of their design.

Stakeholder-centric – A number of use cases for architectural knowledge can be regarded as Stakeholder-centric. These use cases involve identifying stakeholders and communicating the architecture towards these stakeholders. Five out of the seven use cases in this cluster are regarded as important by the respondents. Especially the High-level role deems these use cases important. The remaining use cases ‘identify affected stakeholders on change’ and ‘identify key architectural decisions for a specific stakeholder’ are deemed neutral. Furthermore, stakeholder-centric use cases are regarded as more important at the architecture levels Enterprise architecture and Process and information architecture than at the other levels. This confirms the general idea that the architecture levels Enterprise architecture and Process and information architecture are suitable for communicating architecture to non-IT stakeholders. The other way around, practitioners engaged in Software architecture and Systems architecture do not regard communication of the architecture to stakeholders as important. Apparently, at these more technically oriented levels of architecture, practitioners mainly capture architectural decisions for themselves and not for communication to other stakeholders. This in itself is not bad, but reveals that different communication needs exist for different architecture levels.

Forward architecting – Four use cases for architectural knowledge fall into the cluster Forward architecting. When we regard the use cases in this cluster we see that ‘add an architectural decision’ is deemed important at all architecture levels and by most architectural roles (only the Specialist role does not regard this use case as important). The use case ‘remove consequences of a cancelled decision’ is not deemed very important. We can identify two reasons for this. First, this use case requires that a practitioner is able to cancel an architectural decision. Consequently, the practitioner should determine the decision that needs to be cancelled. This requires the practitioner to make a review iteration. Second, this use case does not directly contribute to the forward-engineering paradigm we identified when we analysed the Assessment use cases. Other use cases in this cluster, such as ‘reuse architectural decisions’ and ‘retrieve an architectural decision’ are deemed important by all architectural roles and at all architecture levels. These results show that the practitioners regard architectural decisions as an important asset to be reused in developing a specific architecture.
In addition to the results listed in Table 2.5, we make another observation. A difference exists with respect to the perceived importance of use cases between the clusters *Communicator*, *Low-level*, and *Specialist* on the one side, and *High-level* on the other side. The cluster *High-level* regards more clusters of use cases important than the other clusters. A possible reason lies in the fact that practitioners in the *High-level* cluster have a wider perspective on architecture and stakeholders involved, whereas practitioners in the other clusters have a more narrowed focus on architecture. This corresponds with the variety of roles and activities of a software architect listed in *(Hofmeister et al., 2000)*.

### 2.5 Threats to Validity

We describe the threats that our case study faced according to *(Kitchenham and Pfleeger, 2001-2002; Kitchenham et al., 2002; Runeson and Höst, 2009)*. Our survey was targeted at practitioners in the Netherlands. By carefully selecting the participants for the survey, we have attempted to minimize a selection bias. Nevertheless, IT service providers are somewhat overrepresented in our population. However, when we compared the responses of practitioners employed at IT service providers with those of practitioners employed at other organizations we did not find significant differences. This strengthens our idea that the construct validity of the survey instrument is adequate.

We controlled the population of practitioners we invited to participate in the survey. However, we do not have insight into the reasons why the non-respondents did not participate. We conjecture that these practitioners did not have enough time to administer the survey or could not relate the topic of the survey to their daily work. Although our survey satisfies the guidelines for the number of questions and maximum administration time as posed in *(Kitchenham and Pfleeger, 2001-2002)*, our results may suffer from a maturation effect, which means that the attitude of the participants towards the use cases in the survey changes *during* filling in the survey. On the one hand, use cases in the first half of the survey receive a more important rating than use cases in the second half. On the other hand, the second half does contain several use cases rated ‘important’. Therefore, we have confidence that the maturation effect did not influence our results substantially.

It was not possible to obtain a structure in the use cases for architectural knowledge based on the practitioners’ answers alone. Apparently, the survey answers varied too much to be used for structuring the use cases. A reason for this could be that our study is based on *more recent definitions of architecture as made of a set of architectural decisions* *(Jansen and Bosch, 2005; Kruchten et al., 2006; Rozanski and Woods, 2005)*. Some participants may regard architecture as a set of components and connectors and are not yet used to viewing architecture as a set of architectural decisions and their rationale. Our approach, which uses a list of use cases for architectural knowledge, may have biased the results since the actual mindset of architects may require additional use cases or other approaches to be fully captured. We provide an architectural knowledge-oriented view towards the mindset. In summary, these factors may have influenced the
To be able to reflect on the answers given, we identified a clustering based on the use cases for architectural knowledge alone and related the answers to these clusters. The resulting reflection in §2.6 is not only based on the clusters of use cases, but puts the survey results in a broader perspective.

2.6 Discussion and Conclusions

We conducted survey-based research on how the practitioners in software architecture in the Netherlands view and use architectural knowledge. Our results reveal the importance of certain use cases for architectural knowledge for the daily work of the practitioners. The individual results have been discussed in §2.4.5. This section reflects on these results and draws overall conclusions on the architect’s mindset and the role of architectural knowledge in that mindset.

Fig. 2.1 provides an overview of the results and depicts the major elements of the reflection. We approach architecture from two different perspectives. One perspective is focused on developing a solution, i.e., the architecture. The other perspective is focused on the underlying reason for that solution, i.e., architectural decisions and rationale. The clusters of use cases for architectural knowledge are depicted as package symbols. The + -mark or − -mark indicate the respondents’ view on these clusters. We put the clusters in perspective by depicting the evolution between the different results that we identify in practice. By and large, widespread acceptance of architecture verification activities preceded architecture validation activities, such as performing risk or trade-off analyses. Similarly, viewing architecting as a forward decision-making process preceded managing the set of architectural decisions, i.e., architectural knowledge management.

Putting stakeholders central in architecture has been an important characteristic across time and perspectives. The remainder of this section describes our views as expressed in Fig. 2.1.

Figure 2.1: Overview of the architect’s mindset
2.6. Discussion and Conclusions

Forward architecting – Architects regard taking architectural decisions and making these decisions explicit as important. Yet, architects tend to focus on only taking architectural decisions to end up with a correct software architecture for a specific problem. In taking these decisions, architects are supported by e.g., architectural patterns (Buschmann et al., 1996), which provide proven architectural solution fragments for certain problems, and by rationale tools such as gIBIS (Conklin and Begeman, 1988) and QOC (MacLean et al., 1996). We signal an ongoing tension between making architectural decisions and capturing the underlying rationale and other context of these decisions; the time spent on capturing the context is not spent on making new architectural decisions. Consequently, adequate, lightweight tooling is necessary to lower the threshold for capturing the context. Despite the continual tension, progress has been made (Fischer et al., 1996; Conklin et al., 2001).

Architectural decision set – On a more generic level, architects do not regard the architecture as a set of architectural decisions. Although the concept of architectural decisions in itself has gained importance, the architect’s mindset lacks focus on reflections on those decisions as building blocks for software architectures. These reflections allow for a step back to actually learn from architecture experiences. Furthermore, architects do not (yet) manage or manipulate that set of architectural decisions (i.e., use cases in the cluster Architectural decision set). A reason for this could be that more recent definitions of software architecture in terms of architectural decisions (Jansen and Bosch, 2005; Kruchten et al., 2006; Rozanski and Woods, 2005) are not yet completely transferred to practice. In addition, adequate tool support is necessary to fully exploit architectural knowledge as a set of architectural design decisions across architectures and domains. This thesis provides more information on the topic of tool support for architectural knowledge management in §1.2.7 and Chapter 8.

Assessment – reqs.→arch.→impl. – Software development largely occurs via projects. Depending on the development approach chosen, the architecting phase can run in parallel during the lifetime of the project or the architecting phase is a distinct phase which leads to a deliverable – the architecture. Based on the results of this study, we conjecture that the latter is the case: the practitioners show an approach in which the architecture is delivered based on the requirements. After that, the implementation is checked against the architecture. Our experience shows that this verification phase often is not performed by architects. Architects, often experienced and relatively expensive practitioners, perhaps run off to another project to run the architecting phase at that project. Consequently, they may not be offered the time to support the design and implementation phase.

Assessment – risk, trade-off analysis – Our study shows that methods and techniques to validate the architecture, such as the Architecture Tradeoff Analysis Method (Clements et al., 2001) or their predecessors, are not embedded within the mindset of architects. A recent presentation on the topic of this chapter given during the Dutch architecture conference revealed that when practitioners do deem performing a risk analysis important, they do not have clear what the role of architectural knowledge is in a risk analysis. Architectural knowledge may support to evaluate the impact of architectural decisions on the resulting architecture; it allows to (re-)consider alternative decisions.
2. The Mindset of Architects

as well. Apparently, this rather new view on architecture is not yet generally accepted. Education on viewing architecture as architectural decisions (Smolander, 2002) as part of architectural knowledge could help overcome this.

**Stakeholder-centric** Another benefit of architecture is that it enables communication among stakeholders (Bass et al., 2003). Architecture thus can be regarded as a language to transfer the architect’s opinions and views to those stakeholders. Most use cases in the cluster Stakeholder-centric rate high, which means that the view of ‘architecture as language’ (Smolander, 2002) is generally accepted. Communication of architecture to stakeholders is clearly established in the mindset of architects.

Our study shows that the mindset of architects is focused on delivering a solution and capturing the related architectural decisions. Consequently, we conjecture that a so-called micro view on software architecture largely is in place: architects are focused on developing an architecture for a specific solution and (more and more) on capturing the architectural decisions and rationale for that solution. What lacks in the mindset of architects is a view that exceeds specific architectures but puts architectures in context by validating them, and the architectural decisions that led to them. When architects have a set of architectural decisions at their disposal, this offers the opportunity to interrelate architectural decisions taken in the past to identify learning opportunities for future architecting activities. We conjecture that this macro view may be achieved by applying initiatives that proved valuable in other disciplines, such as ontology engineering (Gruber, 1993; Noy and McGuinness, 2001) onto the domain of (software) architecture.

In summary, the mindset of architects in the Netherlands reveals an approach which is focused on ‘to create and communicate’ rather than ‘to review and maintain’. This reflects a general pattern as e.g., highlighted in (Tang et al., 2006). Furthermore, architectural knowledge and the view of architecture as a set of architectural decisions has not yet transferred to industry. Kruchten (2008) discusses a similar balance between an external focus (aimed at outward and inward stakeholders) and an internal focus (aimed at taking the right design decisions, validating them, and documenting them); based on our case study, we conclude that the external focus largely is in place, but the internal focus needs more emphasis. We see two possible approaches to embed the importance of architectural knowledge and design decisions in industry. First, more knowledge transfer is needed on the concepts and intended benefits of this view. Second, it is necessary to collect more empirical data on these benefits in terms of throughput and cost to fully sustain the importance of architectural knowledge and architectural decisions.

2.7 Future Work

This chapter describes the mindset of architects in the Netherlands. We provided several reasons for this mindset but acknowledge that additional research is needed on the foundation for this mindset. This additional research could focus on the activities needed to effectively establish the concept of architectural knowledge in the architect’s mindset. The possible increase in understanding of architectural knowledge by architects may be monitored by using our survey instrument periodically. Moreover, it is possible to
compare the mindset of architects in the Netherlands with the mindset of architects at other countries or continents by reusing this survey.

We envision the use cases for architectural knowledge to define operations on a grid for architectural knowledge. We view this grid to support satisfying the need for architectural knowledge from different perspectives. A model that lies at the basis for this knowledge grid and supports capturing architectural knowledge is provided in (de Boer et al., 2006). A further exploration of this grid is given in (Lago et al., 2010).

Within our research project, we are developing, notations, tools, and associated methods to extract, represent, and use architectural knowledge. This chapter has shed light onto the most important use cases for architectural knowledge from a practitioners’ perspective. Although specialized tool support for the architects is still generally lacking, we use these results to develop tools for the most important use cases for architectural knowledge (see Chapter 8). In addition, we continue the work in our project to further embed the view of architectural knowledge and architectural decisions in practice.
This chapter describes a method for assessing organizations involved in global software development to determine deficiencies in securing compliance with architectural knowledge. We report on a case study in which we applied the method in a large organization involved in global software development. The case study reveals that, while initially problems were expected with regard to the structure of the architectural rules, the real problems pertain to the process by which the architectural rules are captured, disseminated, and secured within the organization. Several recommendations are provided to improve this process.

3.1 Introduction

An increasing number of organizations applies offshore software development in order to deliver solutions to its customers. This kind of global software development faces additional problems compared to collocated development, such as differences in culture and spreaded software development activities. This spread requires additional communication and coordination efforts. These efforts aim to guarantee successful integration of subsystems into a working software product that meets the requirements.

Aforementioned communication and coordination efforts can be addressed by issuing rules for the software architecture. Architectural rules are the principles and statements on the software architecture that must hold at all times, and thus must be complied with. Architectural rules are not limited to software artifacts only, but can include organizational processes that manipulate or create the artifacts as well. For example, a rule on branching might read: “a subsystem is the unit of branching, i.e., a subsystem branch is a branch of all assets in the subsystem”.

When an organization has a set of architectural rules installed, these rules act as (additional) non-functional requirements to the software architecture. Lack of compliance with architectural rules often results in architectural mismatch [Garlan et al.].
causing problems when reusing or integrating certain software elements from different sites. Examples of these problems include an error-prone construction process or excessive code size.

For an organization to comply with architectural rules, the rules need to be enforced. Besides enforcing the architectural rules themselves, achieving compliance requires guidelines for how to disseminate and secure the rules within the different development sites.

Differences in culture and a spread across different locations hamper compliance with architectural rules at organizations involved in GSD. This puts pressure on the guidelines targeted at disseminating and securing the rules. In order to bring the use and effectiveness of these guidelines to the surface, an assessment of the way these GSD organizations secure their architectural rules may prove valuable.

This chapter describes a method for assessing organizations involved in GSD to determine deficiencies in securing architectural compliance. We report a case study in which we applied the method in a large organization in global software development, Organization A. The case study revealed that, while initially problems were expected with regard to the structure of the architectural rules, the real problems pertained to the process by which the architectural rules are captured, disseminated, and secured within Organization A.

This chapter is structured as follows. In §3.2 we describe related work on architectural rules and methods to evaluate software architectures. Next, §3.3 introduces Organization A, where we performed the case study: a large organization involved in GSD in the consumer electronics domain. The method used and the experiences in applying the method in the organization is described in §3.4. Next, §3.5 describes the results of the assessment and the subsequent steps the organization took. Finally, §3.6 lists our conclusions.

### 3.2 Related Work

Recent work regards software architecture as a set of architectural design decisions (Jansen and Bosch, 2005; Jansen, 2008). Capturing architectural design decisions enables more architectural knowledge to be represented explicitly in the software architecture and thus prevents vaporization of architectural knowledge. Architectural knowledge consists of the set of architectural design decisions and the resulting design (Kruchten et al., 2006). A specific subset of the architectural knowledge is formed by architectural rules. The need for architectural rules to guide and constrain the amount of ‘artistry’ in software architecture was initially identified in (Boasson, 1995).

Architectural rules are those architectural design decisions that need to be complied with throughout the organization. Architectural rules can be defined using a decision view on software architecture as described in (Smolander, 2002).

Over the past few years, methods for assessing and evaluating software architectures (Clements et al., 2001; Obbink et al., 2002) have been developed. The steps in the method we followed fit the general guidelines on how to conduct architectural reviews as described in the SARA report (Obbink et al., 2002).
3.3. Case Study Description

Herbsleb et al. (2001) report on a large case study in which global software development was compared to collocated development. Their contribution shows that GSD lowers the degree of informal communication and consequently the level of insight a site has into the software developed at another site. GSD lacks a certain amount of trust provided by informal communication and needs a more formal communication process. Furthermore, according to (Ovaska et al., 2003), coordination of GSD can be organized according to standardized processes and written specifications. Our work regards software architecture as a coordination mechanism since the architecture divides a system into components that can be developed relatively autonomously. The processes and specifications that guide this development form what we regard as architectural rules. Compliance with these rules is of paramount importance for GSD organizations. However, achieving compliance requires additional effort and guidelines. In this chapter, we report on our experience in assessing an organization involved in GSD for these efforts and guidelines.

3.3 Case Study Description

We performed a case study at Organization A, a large software development organization that has multiple sites where software is being developed for a consumer electronics product. Software development within this organization is structured in projects. Each project delivers a new type of the product. As such, each project develops software to satisfy the requirements that hold for that product type.

To efficiently deliver the software for these product types, Organization A has developed a product-line architecture. The product-line architecture distinguishes several subsystems, each focusing on specific functionality, such as signal processing or the menu structure of the device. Subsystems are organized into layers. This is but a logical organization of the subsystems into manageable chunks. The relation between projects, layers, and subsystems is depicted in Fig. 3.1.

Subsystems are developed by dedicated subsystem development teams, headed by a project manager and consisting of an architect, a configuration manager, and one or more software engineers. Each subsystem development team is located at a single site. Together, this organization into subsystems and the location of a subsystem development team at a specific site result in a certain amount of autonomy of the subsystem development teams. In order not to let this autonomy lead to problems with integration of software from multiple subsystems, an architecture team is responsible for maintaining the product-line architecture. The architecture team consists of the chief architect, representatives of the major projects that are running, and some main subsystem architects. Most members of the architecture team are located at one particular development site. To maintain the architecture, the architecture team issues architectural rules in small, text-based documents called architectural notes, or archnotes for short. Each archnote addresses a coherent set of architectural rules, e.g., all rules that have to do with naming conventions are captured in a single archnote. In total, 53 archnotes exist containing some hundreds of architectural rules.

Archnotes are mainly targeted at engineers, subsystem architects, and configura-
3. AKM Practices for Compliance in GSD

Figure 3.1: Projects versus subsystems at Organization A. Subsystems in bold deliver software for use in more than one type

...
3.4 Performing the Assessment

We performed an assessment of the way compliance is secured with architectural rules to verify the aforementioned hypothesis. We used the assessment results to identify improvement points for the structure and use of the archnotes.

3.4 Performing the Assessment

This section describes the phases of the assessment method as well as the results of applying the method in the case study. We used an assessment method that helps to reveal problems that exist in securing architectural compliance.

In terms of the SARA report (Obbink et al., 2002), the objective of our assessment was to identify opportunities for improvement in architectural compliance. This objective corresponds with one of the concerns of the chief architect. The preparation of the assessment focused on getting clear the goal of the assessment (reveal problems that exist in securing architectural compliance), identifying the scope (the architectural rules), and selecting participants for the assessment (global architects and important roles in the subsystem development team, such as subsystem architect, engineer, and configuration manager). The assessment activities themselves focused not so much on the architecture itself, but rather on the guidelines and rules in place for the architecture. In observing the guidelines and rules, we specifically regarded both the structure of the architectural rules and the use of the architectural rules. The assessment was concluded by presentations of the assessment results (conclusions and improvement suggestions) to the line management of Organization A and several subsystem development team representatives.

We provide a general overview of the method. The next subsections describe each phase of the assessment method in more detail. In the next subsections, results from the case study are described in italics.

Fig. 3.2 provides an overview of the assessment activities:

1. Based on the objective of the assessment, our analysis starts with determining and analyzing the audience (population) of the architectural rules. This results in insight into the usage of the architectural rules at different sites.

2. Based on insight into the audience of the architectural rules, we construct a questionnaire. The questionnaire contains specific questions on the following topics: the relevance of architectural rules, the perception of participants of the role of architectural knowledge within the organization, and the participants’ opinion of the architecture team. These questions aim to provide information on the hypotheses formulated in §3.3. Next, we send out the questionnaire to a selection of employees which includes at a minimum all major roles involved at the major development sites. We analyze the answers to the questionnaire to find e.g., conflicting statements and to find specific suggestions as to where to dig deeper.

3. Using the same selection criteria as with the questionnaire, we select a subset from the questionnaire participants to conduct interviews with. We use the interviews to foster specific suggestions for removing deficiencies in securing
architectural compliance. The questionnaires and interviews may reveal potential issues in securing architectural compliance. For that reason, we send out the questionnaires to participants individually and conduct individual interviews (Obbink et al., 2002).

4. The suggestions collected in the interviews in turn are used to provide recommendations to the software development management of Organization A, including team leaders and product management.

5. Insight into what architectural rules are most important for the participants’ daily work enables us to study these rules. We can determine the structure that seems most appropriate to disseminate architectural rules. This may result in an improved classification of the architectural rules. Finally, the architecture team uses the list of most important architectural rules to see if these rules have a higher relevance and require a different way of securing compliance with them.

3.4.1 Analysis of the population

The potential population that has to comply with the architectural rules consists of all employees that are engaged in the architecture for which the architectural rules are compulsory.

The actual usage of the architectural rules across development sites provides information on that part of the population that actually reads them. We distinguish between different sites to see whether differences exist between those sites.

Organization A develops its software using four main development sites. Each site has a main configuration manager role assigned. One of the tasks of the main configuration manager is to grants employees access to the source code tree. We contacted the
main configuration manager of each site to obtain the number of employees involved in software development that reside at that site.

We used the log files of the intranet-site web server of the archnotes to get insight into the actual usage of archnotes. The analysis spans the period from January, 2005 until April, 2006. Fig. 3.3 shows the usage of archnotes at the four main development sites. The total number of employees involved is 515. Of these potential users of archnotes, 288 have accessed at least one archnote.

![Figure 3.3: Usage of archnotes per site](image)

Although the number of potential users is roughly the same for the two major development sites (sites A and B), significant differences can be observed in the actual number of users from those sites that access the archnotes. The site with the highest ratio of potential versus actual users (site A) is the site at which the architecture team resides.

The actual usage of the architectural rules provides information on the number of queries (‘hits’) on architectural rules from different sites. This helps in identifying what kinds of architectural rules are accessed most often. In the next step of the method, among other things, we determine if these architectural rules are indeed the architectural rules deemed most important for the participants’ daily work (and thus, whether existing rules satisfy the knowledge need).

Using the same log files, we counted the number of hits on archnotes. To prevent a bias, we regard multiple queries of a single user to a single archnote file within a period of two minutes as one hit. This two-minute threshold was chosen after a discussion with several architects – they indicated that every user of an archnote should be able to find the information in it within two minutes.

We wanted to know which archnotes were accessed more often than others and distinguish between development sites. To this end, we plotted the number of hits per development site for the archnotes accessed most often – see Fig. 3.4. The usage of archnotes...
3. AKM Practices for Compliance in GSD

![Graph showing number of hits per archnote per development site.]

*Figure 3.4: Number of hits per archnote per development site*

provided valuable information as to what knowledge of the software architecture is accessed most often. Archnotes accessed most often pertain to software artifacts (such as subsystems, naming conventions, and branching of subsystems) and coding standards. These topics are typically covered in the development view or code view as covered by [Kruchten, 1995] and [Soni et al., 1995], respectively.

We used the analysis of the audience of the archnotes in two ways:

- To select participants for questionnaire-guided interviews to obtain insight into the actual status of securing architectural compliance and suggestions on how architectural compliance can be improved within Organization A.

- To determine whether archnotes that are accessed more often are also regarded to be more important to inform of architectural rules.

Analysis of the intranet website usage does not provide an undisputed view on the actual usage of architectural rules throughout Organization A: other mechanisms besides accessing a repository can be used to share architectural knowledge, such as informal chats and team presentations. In the interviews we verify the importance of the intranet website in disseminating architectural rules.

---

1Especially the archnotes on subsystem lifecycle, describing the possible states of a subsystem, and subsystem releases, describing the process and conventions used when making a release of a subsystem for integration purposes, are important in GSD organizations. Compliance with these archnotes enables efficient integration of subsystems.
3.4. Performing the Assessment

3.4.2 Conduct questionnaires

The next phase of our assessment method starts with developing a questionnaire based on the results collected so far. We use a questionnaire to gain understanding of the underlying reasons for the usage of the architectural rules. Furthermore, we try to obtain evidence to support our hypotheses stated earlier.

The questionnaire consists of the following elements:

1. Demographic data – Obtain name, role, years of experience, and subsystem experience of the participant.
2. Knowledge sharing – Identify practices and supporting mechanisms used by the participant to share knowledge, architectural knowledge in particular.
3. Opinion on the architecture team – Identify the participant’s opinion on the role of the architecture team and the activities the architecture team undertakes to disseminate architectural knowledge and secure compliance.
4. Importance of architectural rules – Identify the importance of the architectural rules for the daily activities of the participant. In this step, we tailor the questionnaire to contain the actual list of architectural rules that are applicable to Organization A.
5. Missing architectural rules – Identify what topics exist that require architectural rules, but do not have them.

The questionnaire is designed to obtain feedback from the assessment participants on each of the topics mentioned above. This feedback results in information on both the architectural rules themselves as well as the activities that are in place within the organization to disseminate and secure the architectural rules. In addition, the questionnaire allows participants to bring forward suggestions for improving the current situation.

Based on the analysis of the audience of the archnotes we identified a representative subset of 20 employees. These employees include the most important roles from the four main development sites. We sent out the questionnaire to these employees and obtained the following results:

1. We included 15 participants in either the subsystem architect or engineer roles. The remainder of the participants were 2 configuration managers, an event manager, a quality assurance officer, and a requirements manager. The participants covered 58% of all subsystems and three of the four main development sites. We selected 9 participants from site A, 9 from site B, and 2 from site C.
2. Table [41] shows the use of knowledge sharing practices at Organization A. We found that the most important mechanism to share knowledge is the formal mechanism of a change request. A change request is issued using the configuration management system. The importance of other mechanisms for knowledge sharing is significantly lower than the importance of a change request. These other
3. AKM Practices for Compliance in GSD

mechanisms all are less formal than using a change request. Although these informal mechanisms are important in general, they are applied less often due to the nature of the GSD organization. This has also been reported in Ovaska et al. (2003).

3. All participants regarded the role of the architecture team as pivotal for disseminating the architectural rules. Table 3.3 lists the opinion of the participants on the architecture team. The table shows the degree to which the participants agree with the statement. We observed the three statements with the lowest score (indicated in boldface in the table):

(a) participants felt that they were less regularly updated on the subjects addressed by the architecture team.

(b) participants indicated an improvement point in the response time of the architecture team on a specific request for the team issued directly or by delegation.

(c) participants felt that the architecture team can improve on the point of being in control of architectural activities. We investigated these reasons in order to foster suggestions for improvement during the follow-up interviews.

4. We asked the participants to indicate the ten most important archnotes for their daily activities. Table 3.2 shows the results. We see that most of the important archnotes are the ones that were accessed most often using the intranet website (see Fig. 3.4). Consequently, we concluded that the intranet website serves as an adequate mechanism to disseminate the architectural rules.

5. As a final result of the questionnaire, we collected a list of missing architectural rules. Missing architectural rules concern architectural topics for which architectural rules are requested, but not present. The participants indicated to need architectural rules for the following topics:

- Dynamic aspects (execution, memory consumption, inter-process calls) of the software architecture.
- Deadlocks and preventing deadlocks in software.
- Dealing with state changes in an end product.
- Integration strategies of different subsystems.

These architectural topics pertain to dynamic aspects of the software architecture. Addressing these aspects may be done by introducing a process view according to Kruchten (1995). In the absence of architectural rules for these topics, the knowledge is currently obtained through informal communication. Although informal communication is important, it reduces traceability of compliance with these rules.
3.4. Performing the Assessment

Table 3.1: Practices for knowledge sharing

<table>
<thead>
<tr>
<th>Cluster label</th>
<th>Levels of architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Request using CM</td>
<td>12</td>
</tr>
<tr>
<td>Informal chats</td>
<td>7</td>
</tr>
<tr>
<td>(Group) presentations</td>
<td>7</td>
</tr>
<tr>
<td>Teleconference</td>
<td>6</td>
</tr>
<tr>
<td>(Face-to-face) meeting</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 3.2: Most important archnotes

<table>
<thead>
<tr>
<th>Archnote subject</th>
<th>Number of votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming conventions</td>
<td>12</td>
</tr>
<tr>
<td>Subsystem branching</td>
<td>10</td>
</tr>
<tr>
<td>The concept of a subsystem</td>
<td>9</td>
</tr>
<tr>
<td>Overview of subsystems</td>
<td>9</td>
</tr>
<tr>
<td>Subsystem directories</td>
<td>8</td>
</tr>
<tr>
<td>Subsystem releases</td>
<td>7</td>
</tr>
<tr>
<td>Top level directories</td>
<td>6</td>
</tr>
<tr>
<td>Subsystems in layer 1</td>
<td>6</td>
</tr>
</tbody>
</table>

The questionnaire results show no difference in opinion at different development sites on important knowledge sharing mechanisms, the architecture team, or the most important archnotes. We already observed a low usage rate from site B, compared to the potential number of users of archnotes. We attributed this to the fact that the architecture team resides at site A. Employees at site B received notifications of new architectural rules in the form of a new or updated archnote. Consequently, site B might regard the archnotes as ‘not-invented-here’ and has a lower urge to read them and comply with the architectural rules. We used follow-up interviews to validate this hypothesis.

3.4.3 Conduct follow-up interviews

The questionnaire results are collected and analyzed to formulate additional questions for the interviews. Possible questions arise from contradictions in the answers of participants and contradictions between the questionnaire response and the analysis of the population (see §3.4.1). In order to obtain unbiased feedback from the participants, the interviews are held with the participants individually. The interview results were studied and combined to determine overall improvement points.

We interviewed seventeen interviewees from the list of participants. In case multiple employees work on the same subsystem in the same role, we selected only one of these employees. While conducting the interviews, we specifically focused on the role of the architecture team to find reasons for the scores listed in Table 3.3. Studying the interview results led to insight into the reasons for the scores and suggestions to improve the
3. AKM Practices for Compliance in GSD

Table 3.3: Opinion on the architecture team

<table>
<thead>
<tr>
<th>Opinion</th>
<th>% agreed</th>
</tr>
</thead>
<tbody>
<tr>
<td>The goal of the architecture team is clear to me.</td>
<td>71.43%</td>
</tr>
<tr>
<td>I am regularly updated on the subjects addressed by the architecture</td>
<td>52.86%</td>
</tr>
<tr>
<td>team.</td>
<td></td>
</tr>
<tr>
<td>I know where to find the architecture team’s information.</td>
<td>80.00%</td>
</tr>
<tr>
<td>Decisions are properly made by the architecture team.</td>
<td>67.14%</td>
</tr>
<tr>
<td>Decisions are properly documented and deployed by the architecture</td>
<td>67.14%</td>
</tr>
<tr>
<td>team.</td>
<td></td>
</tr>
<tr>
<td>The response time of the architecture team is OK.</td>
<td>56.92%</td>
</tr>
<tr>
<td>The architecture team is sufficiently in control of architectural and</td>
<td>60.00%</td>
</tr>
<tr>
<td>design activities.</td>
<td></td>
</tr>
<tr>
<td>The representation in the architecture team is effective.</td>
<td>68.57%</td>
</tr>
<tr>
<td>I am aware of the existence of the reference architecture (see Fig. 3.1</td>
<td>82.35%</td>
</tr>
<tr>
<td>as an indication).</td>
<td></td>
</tr>
<tr>
<td>The purpose and content of the reference architecture is clear to me.</td>
<td>78.57%</td>
</tr>
<tr>
<td>The content of the reference architecture is useful to me.</td>
<td>80.00%</td>
</tr>
<tr>
<td>I am aware of the architecture conventions (archnotes).</td>
<td>94.12%</td>
</tr>
<tr>
<td>The purpose and content of the archnotes is clear to me.</td>
<td>84.62%</td>
</tr>
</tbody>
</table>

The interview results offered the following suggestions related to the entries in boldface in Table 3.3:

- Several participants experienced uncertainty on certain architectural topics. The participants see projects deviate from the architectural rules for the sake of the project. The architecture team often participates in discussions and is aware of the deviations. Yet, these deviations are not communicated throughout the development organization. This lack of communication results in the opinion that the architecture team is not in control and trailing on projects.

- The visibility of the architecture team is further reduced by a lack of resources in capturing the architectural rules in archnotes. At the moment, nineteen archnotes have the status ‘planned’ and have not yet been written. Although a list of missing architectural topics was identified, these are not expected to be addressed within a few months.

- The effectiveness of the architecture team’s meetings can be improved. Members of the team that were interviewed indicated that both the current frequency of the meeting and the attendance should be higher. They felt that this would improve the visibility of the architecture team to the rest of the software development community.

Aside from these specific suggestions to improve the role of the architecture team, we identified other problems experienced by the participants. The most important problems
3.4. Performing the Assessment

Table 3.4: Problems that prevent securing architectural rules

<table>
<thead>
<tr>
<th>Category</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use of architectural rules</strong></td>
<td>Projects deviate from terminology in architectural rules</td>
</tr>
<tr>
<td></td>
<td>Projects deviate from architectural rules (way of working)</td>
</tr>
<tr>
<td></td>
<td>Too many architectural rules exist</td>
</tr>
<tr>
<td></td>
<td>Architectural rules are outdated or draft</td>
</tr>
<tr>
<td><strong>Architectural topics</strong></td>
<td>Too many architectural rules cover organizational aspects</td>
</tr>
<tr>
<td></td>
<td>Too many architectural rules do not cover ‘architecture’</td>
</tr>
<tr>
<td></td>
<td>Some architectural rules contain too much text</td>
</tr>
<tr>
<td><strong>Architecture team</strong></td>
<td>The architecture team is trailing on the reality in projects</td>
</tr>
<tr>
<td></td>
<td>Compliance with architectural rules is not verified</td>
</tr>
</tbody>
</table>

that prevent the organization from securing the architectural rules at different development sites are listed in Table 3.4.

At this point, our initial hypothesis that the problem for securing the architectural compliance was in the structure of the archnotes seemed to be wrong; all problems identified pertain to the topics of archnotes, the deviations from archnotes, and the lack of verification of compliance with archnotes which prevents securing architectural rules within Organization A. These problems concern the process by which the architectural rules are captured, disseminated, and secured within the organization.

3.4.4 Provide recommendations

We used the interviews to collect suggestions on how to improve architectural compliance. In analyzing the compliance topics in the questionnaire, we focused on those answers that indicated that room for improvement was possible according to the participant. Next, we asked the participant how, from the participant’s perspective, this improvement would be most expedient. The collected responses were grouped to form suggestions. The five suggestions mentioned most often by the interview participants are listed below:

- Provide an email notification of changed or new archnotes to subscribers.
- Let the architecture team prioritize the archnotes according to relevance.
- Let projects write down deviations from archnotes.
- Let the architecture team verify compliance with archnotes.
- Let subsystem architects periodically discuss archnotes in their team meetings.

These suggestions were presented to Organization A’s management to obtain support for improvement of securing architectural rules in the organization – see §3.5.
3.4.5 Classify architectural rules

At this point, valuable suggestions are collected from the previous steps of the assessment. Nevertheless, an analysis of the structure of the architectural rules can be performed in order to reveal additional suggestions. This analysis starts with those architectural rules that were regarded as most important for the participants’ daily work. Expliciting the structure of these rules and applying this structure to other architectural rules is expected to improve the readability of the architectural rules. This in turn would improve dissemination of the architectural rules within the GSD organization.

In expliciting the structure of the architectural rules, the concept of architectural views is used. Architectural views (Clements et al., 2003) prove valuable in structuring architectural knowledge, such as architectural rules.

During this step, the existing architectural rules are analyzed to determine whether they are structured using architectural views. If this is not the case, existing view models such as the ones described in (Kruchten, 1995; Soni et al., 1995; Clements et al., 2003) can be recommended.

Based on a study of the most important archnotes, we observed that they contain a variety of rules pertaining to the architecture. Furthermore, the architectural rules serve as an aid for multiple stakeholders. For example, an engineer would like to learn how the subsystem’s code tree is organized to locate software assets. On the other hand, the configuration manager would like to learn how the code tree is packaged into a subsystem release to verify that a subsystem release is made correctly. Currently, the archnotes do not distinguish between stakeholders and their concerns. This hampers a stakeholder in finding the right information in the archnotes. Therefore, we recommended a classification of the architectural rules. A classification would improve finding the right architectural rules and, consequently, improve complying with these rules. We proposed a classification in two ways:

- A specific architectural view. At the moment, the archnotes are not structured according to views. The architecture team could use the most important archnotes as perceived by the audience of the archnotes as a starting point. Next, the architecture team could reclassify the architectural knowledge in the archnotes and indicate what knowledge is of particular interest for what role from a compliance perspective. We already learned that the dynamic aspects of the software architecture are underexposed by the architectural rules. Specific attention could be directed towards closing this gap by reverting to the process view described in (Kruchten, 1995).

- A classification according to importance of compliance with these rules. The architecture team could indicate the relevance for compliance with architectural rules. Possible suggestions to discriminate in relevance include cost of non-compliance (“what will non-compliance with this rule cost me?”) or interoperability (“what subsystems need to be changed when this architectural rule is issued?”).

Organization A regarded these suggestions as helpful to improve securing the archnotes. However, most of the suggestions of the participants done so far pertained to the
process by which archnotes are created and disseminated. We concluded that the organization deems these process-related suggestions as more feasible to achieve improved compliance with the archnotes.

Introducing views in archnotes improves the structure since target stakeholders and concerns are explicitly addressed – this may improve the use of the archnotes. Moreover, formalizing the architectural rules in the archnotes enables compliance checking, which would be an incentive to use the archnotes as well. We also observed in this case study that formalized mechanisms such as change requests are successful in use. In conclusion, we hypothesize that a change in formalization of the archnotes could improve the process.

3.5 Results of the Case Study

The case study led to several results. First, Organization A gained insight into the potential audience of architectural rules and the average usage of the rules. Second, questionnaire analysis and interviews revealed that the main problem with securing architectural rules within Organization A was not so much in the rules themselves, or in the structure used to capture them. Rather, the real problems pertained to the process by which the rules are captured, disseminated, and compliance to the rules is secured within the organization.

In order to overcome these problems, we collected five specific improvement suggestions from the interviews. Next, these suggestions were presented to the management of the software development organization. After a period of reflection, the management decided to support four out of five improvement actions. Only the suggestion to send out change notifications of architectural rules was not supported, because management thought that this would result in an overloading of electronic communication. The supported actions all aim to improve the process by which architectural rules are used within Organization A and do not so much concern the way in which architectural rules are structured. Below, we list the four supported improvement actions:

**The architecture team prioritizes the archnotes according to relevance** – The priority is based on the architecture team’s knowledge of the archnotes themselves and the insight gained from this case study. Difference in relevance is supported by the results of the interviews in that some archnotes do not discuss what the organization would call “architecture”, and that architectural rules on certain topics are missing. Furthermore, this prioritization offers Organization A the possibility to tailor e.g., the dissemination process or compliance verification process of a specific archnote.

**Architects periodically discuss the archnotes in (subsystem development) team meetings** – Although the fact that archnotes exist is mentioned in subsystem development team meetings, the contents of archnotes are not discussed. This relatively little attention does not result in engineers paying attention to the rules. Addressing and explaining the rules instead is expected to increase the awareness of the rationale behind certain rules which is a prerequisite for compliance – it is expected to reduce the not-invented-here-syndrome. Explaining why a rule is the way it is (Tyree and Akerman, 55
3. AKM Practices for Compliance in GSD

In addition, we suggest that the architecture team uses techniques such as ‘traveling architects’ (Corry et al., 2006) and visits other development sites more regularly. Visiting a development site and discussing archnotes helps to increase the number of employees from that site that read and understand the archnotes.

Projects write down deviations from architectural rules – Several projects deviate from certain architectural rules for the sake of the project. Often, this fact is known, but not addressed by either changing the rule that was broken or issuing a ‘known deviation’ to the architectural rule. Not communicating this deviation results in a lack of clarity about the authority of the rule. To prevent this, projects should provide a written statement that they deviate from an architectural rule, including the rationale for this deviation. Next, these written statements are sent to the architecture team to obtain approval.

Verify compliance with the architectural rules – Compliance with architectural rules currently is not verified. Not verifying rules that are mandatory leads to a certain amount of indifference with engineers (“it probably is not that important then…”). This indifference is tackled by letting engineers include the architectural rules as explicit criteria in their regular review meetings. In addition, the quality assurance officer verifies that these reviews have been performed correctly. Verifying compliance leads to insight into the extent to which the software that is developed complies with the architectural rules set for it.

3.6 Conclusions

Architectural rules are necessary in guiding, coordinating, and communicating software development efforts across multiple development sites. Architectural rules require additional guidelines on how to disseminate and secure the architectural rules within the different development sites to achieve compliance. This chapter describes a method for assessing organizations involved in GSD to determine deficiencies in securing architectural compliance. Application of the method on a case study showed that an explicit role of the enforcing organizational unit and periodic communication of architectural rules across all sites is of paramount importance. Suggestions for lowering the not-invented-here-syndrome of the architectural rules at the development sites have been identified and are currently implemented.
Reassessing the Need for Practices for Architectural Compliance

In this chapter we extend the research on compliance practices by comparing Organization A and the results obtained during the previous study with a second organization, Organization B, involved in global software development. We describe how each organization uses practices for architectural compliance to overcome challenges innate to GSD. This study shows a difference in the way both organizations use compliance practices; based on a detailed comparison, we conclude that a combination of architectural rules and compliance practices proves valuable in handling GSD challenges.

4.1 Introduction

Global software development faces challenges additional to those of collocated development. The primary obvious difference is that software engineering practices are performed at geographically separate locations. This difference introduces challenges that have been reported in the literature (Herbsleb et al., 2001; Holmström et al., 2006), such as temporal, strategic, and socio-cultural challenges (Ägerfalk et al., 2005). Some promising solutions to address these challenges have been identified (Herbsleb et al., 2005; Mullick et al., 2006), but their real contribution cannot yet be fully distilled from practice because empirical evidence is lacking.

Some of the challenges in GSD originate from the architecture of the software. The architecture can induce dependencies between software engineering tasks such as managing synchronization and meeting a release schedule, see e.g., (Bass et al., 2007b; Conway, 1968). As with collocated development (Boasson, 1995), architectural rules can help to overcome some of these challenges in GSD. Architectural rules are principles and statements about the software architecture that must be complied with throughout the organization (see Chapter 3). We saw in Chapter 3 that where architectural rules concern the architecture as a product, additional measures pertaining to the processes may be necessary for successfully implementing that architecture.

Although using architectural rules seems beneficial for a distributed setting, we lack detailed insight into what kind of architectural rules are generally applied in a GSD en-
4. Reassessing the Need for Practices for Architectural Compliance

In addition, we do not know in what way these architectural rules contribute to overcome challenges specific to GSD.

In this chapter, we provide an overview of the major GSD challenges as mentioned in the literature, and refine them into seven issues. For each issue, we list possible solutions and describe to what extent these solutions can be expressed as architectural rules. Next, we use this overview to study two organizations involved in GSD. We determine what practices are used to overcome the various issues. Our study reveals that architectural rules pertaining to the product (i.e., the structure of the software) and additional measures on the process are relevant. For example, allowed dependencies between subsystems may be defined in architectural rules, but so need the processes to verify compliance with these rules.

This chapter is structured as follows. In §4.2 we provide an overview of related work in the field of software architecture, architectural rules, and global software development. Next, §4.3 provides an overview of the challenges that exist in GSD. This section also addresses possible solutions to overcome these challenges as identified in the literature. In §4.4 we show two organizations overcome these challenges in practice. In the discussion, we focus on the contribution of architectural rules to the handling of GSD issues. Finally, §4.5 lists our conclusions.

4.2 Related Work

This section provides related work in the field of software architecture, the concept of architectural rules, and general challenges identified in global software development literature.

Recently, software architecture is regarded as the set of architectural design decisions (Jansen and Bosch, 2005; Kruchten et al., 2006). Examples of these decisions include preferred or mandatory standards on communication between layers of the architecture or datastructure conventions. Those architectural decisions that must be complied with throughout the organization are defined as architectural rules (Boasson, 1995; Clerc et al., 2007a). Architectural rules can help to overcome challenges in software engineering and software architecture (Boasson, 1995). However, we lack insight into what rules are necessary to guide GSD.

A software architecture imposes coordination and communication efforts between development teams that are necessary to successfully implement the architecture. To minimize these efforts, the software architecture often mimics the structure of the organization: a certain modular architectural design is used as the basis to distribute development work across different teams (Conway, 1968). With GSD, the challenge to distribute work becomes even bigger. As development work is spread across development sites, (Herbsleb and Mockus, 2003) indicate that this demands which demands more coordination and communication effort.

As we have shown in Chapter 3, having a description of the software architecture and rules on the architecture alone is not enough to ensure successful compliance in GSD. These rules should be accompanied by additional process guidelines and activities to let them sink in properly. These additional guidelines and activities pertain to the use
4.3. Challenges in Global Software Development

and personalization of architectural rules as opposed to codification that often receives most emphasis \cite{Hansen2006, Desouza2006}.

Substantial work has been done to provide insight into the problems or challenges that occur when the development effort is spread across multiple development sites \cite{Herbsleb1999a, Herbsleb2005, Herbsleb2001} list several dimensions of the problem with physical separation of software engineering practitioners: technical problems, project management problems, knowledge management problems, and communication problems. Socio-cultural, temporal, and geographical issues are further exemplified in \cite{Agerfalk2005, Holmstrom2006}.

Some solutions to overcome the challenges involved in GSD are also addressed in the literature. \cite{Herbsleb1999a} report on several solutions to overcome the distance innate to GSD, such as attending to Conway’s Law \cite{Conway1968}, traveling at the start of projects to get to know relevant individuals from multiple sites up-front, and recording architectural design decisions including their rationale. \cite{Corry2006} describe a technique in which architects frequently travel between development sites to engage developers in software architecture work. \cite{Bass2006a} proposes an approach in which collaborating practitioners first acquire a mental model of the software engineering task they need to perform \cite{Espinoza2002}, and only then pursue completion of the task. The significance of a shift in perspective is signalled in \cite{Bass2006a}: project management should pay more attention to coordination problems and applied synchronization efforts. Other work of \cite{Bass2006} relates the required communication and coordination efforts in software engineering \cite{Herbsleb2003} to the organization’s capabilities \cite{Bass2007b}. Problems occur when the required communication and coordination efforts exceed the organization’s capabilities. This results in misalignment between the architectural design decisions that have been taken and the organizational structure \cite{Bass2007b}. Since communication and coordination efforts are more important when an organization is involved in GSD, these efforts place additional requirements to the organization’s capabilities.

4.3 Challenges in Global Software Development

In this section, we list four global software development challenges we have distilled from the literature (see Sections 4.3.1 through 4.3.4). We took proceedings from relevant conferences and workshops (the GSD and ICGSE proceedings) and special issues of magazines (e.g., special issues in IEEE Software), both on the topic of GSD, as a basis. We next identified the issues described in these contributions and grouped and related them when possible. Fig. 4.1 provides an overview. For some of the challenges, we made a further subdivision, to finally end up with a list of seven issues. As the figure shows, challenges can amplify other challenges. The remainder of this section describes these challenges, issues, and their interrelations. For each issue, examples of possible solutions are given. We primarily focus on solutions that pertain to the architecture, but mention other solutions when these are not available.
4. Reassessing the Need for Practices for Architectural Compliance

4.3.1 Challenge 1: Time difference and geographical distance

The first challenge in global software development originates from the innate difference with GSD as compared to collocated software development: working at geographical distant sites and the time difference incurred. Geographical distance and time difference burden software engineering activities. Software engineering activities are knowledge-intensive tasks and therefore require communication during various phases in the lifecycle (Herbsleb and Mockus, 2003). Time difference leads to delays because of less overlapping working hours (Herbsleb and Grinter, 1999a). Furthermore, geographical distance leads to delays because it increases the unresponsiveness of practitioners. This challenge amplifies that of culture (Challenge 2), as empirically investigated by (Herbsleb and Mockus, 2003). Differences exist in how this amplification exactly occurs: cultural differences may exist within the same time zone due to large geographical distance (e.g., UK and South-Africa collaborating) and, conversely, cultural similarities may exist between practitioners from countries with a large geographical distance.

4.3.2 Challenge 2: Culture

GSD involves software engineers, software architects, and other practitioners, possibly from all over the globe. People with different cultural backgrounds are required to cooperate in order to deliver working software in time. As shown by (Hofstede, 2001), people with different cultural backgrounds behave differently. We have examined literature on cultural challenges and identified the issue that is regarded as key for supporting software engineering practices in a GSD setting:

1. Difficulty to initiate contact

Establishing contact between practitioners of different cultures faces additional challenges as compared to establishing contact between practitioners of the same culture.
4.3. Challenges in Global Software Development

Moreover, the challenge of culture is amplified when project members located at different development sites want to initiate contact (Herbsleb and Grinter, 1999a). Examples of cultural problems include limitations in the vocabulary of practitioners (Holmström et al., 2006) and difference in communication style: reluctance to ask questions (Herbsleb et al., 2005), direct versus indirect communication (e.g., phone versus email) (Herbsleb and Grinter, 1999b), or giving preference to sending an e-mail over establishing contact directly by phone (Herbsleb and Grinter, 1999a).

Grinter et al. (1999) and Herbsleb et al. (2005) indicate that the main solution to overcome this cultural issue may be to establish face-to-face contact and to get to know ‘who is who’ in the project. Suggestions that can help to overcome this issue are (1) establish a directory listing all practitioners involved in the development of certain parts of the architecture and (2) start all projects with a kick-off meeting. Establishing contact is easier when practitioners are located at the same development site. Once contact has been initiated, practitioners are more willing to overcome their cultural differences in order to communicate effectively.

The cultural challenge described in this section influences or amplifies other GSD challenges, such as communication challenges, collaboration challenges, and work distribution challenges.

4.3.3 Challenge 3: Team communication and collaboration

The nature of global software development results in increased difficulties to communicate and collaborate within teams and between teams involved in software development. We distilled the following issues from literature addressing this challenge:

2. Difficulty to exchange information
3. Difference in sense of urgency
4. Difficulty to build a team
5. No collective ownership

Difficulty to exchange information
Having teams at geographically separated sites results in a lack of unplanned, informal contact during which seemingly irrelevant but in fact highly valuable information is exchanged (Herbsleb and Grinter, 1999a; Herbsleb and Mockus, 2003). The information exchanged during unplanned social meetings is not necessarily architectural in nature, but can help to more easily obtain architectural information. Moreover, unplanned social meetings help people to build awareness of what will happen before a formal decision is made. In particular, awareness of architectural design decisions is important, since these decisions may have a high impact across development sites. In addition, the difficulty to exchange information arises at an early stage in the project, because practitioners just do not know who to contact at a remote site; they simply don’t know the practitioners at a remote site (see Challenge 2).

A number of suggestions to lower the threshold to exchange information across different development sites have been provided in the literature. These suggestions include...
4. Reassessing the Need for Practices for Architectural Compliance

providing incentives to improve collaboration and knowledge sharing. To improve collaboration, teams of experts or communities of practice (Herbsleb et al., 2005), in which gurus can flourish, can be established. Incentives for architectural knowledge sharing include the establishment of social ties and knowledge internalization (Farenhorst et al., 2007c). Other suggestions to address the difficulty of exchanging information include traveling to different development sites at an early stage in the project and meeting various practitioners at these sites. As a result of the face-to-face contact, practitioners experience that there actually is “a person behind an email address” (Herbsleb et al., 2005). This effect can be achieved by processes and policies regulating the frequency, location, and participants of the required inter-site meetings.

**Difference in sense of urgency**

GSD can result in a difference in the sense of urgency across development sites to handle specific requests. A difference in establishing contact on a certain topic (e.g., practitioners prefer to send someone an email over calling that person) influences the sense of urgency to handle that topic. As with the previous issue, getting to know practitioners from other sites is key to overcoming this issue. Once a key individual is already known to practitioners at another site, these practitioners are more willing to pose a question to that individual. This person can even become a liaison, or first contact point for practitioners at the other site (Herbsleb et al., 2001). Communication is sped up, development activities to be performed are agreed upon, and a collective sense of urgency towards the activities that need to be performed is achieved. This collective sense of urgency, in turn, will shorten development time. Measures that focus on the development process can be of value in this case. An architectural rule could make explicit that every project, site, or subsystem should have a liaison. Additional information on how, when, and where to contact this liaison could be provided as well.

**Difficulty to build a team**

Now that we have seen possible solutions to address the difficulties of exchanging information and the difference in sense of urgency, we face another difficulty: it is difficult to build up a single, virtual team across development sites because this is hampered by the geographic distance and time difference (Herbsleb and Mockus, 2003; Dekker, 2008) (see Challenge 1). A single, virtual team is characterized by low communication thresholds and high collaboration possibilities. Not having such a team results in possible mismatches in terminology and definitions and, consequently, communication overhead and delays (Herbsleb et al., 2001).

A possible solution to the issue of building a team is to develop architectural rules that contain conventions and procedures on collaboration and the use of a single, shared environment (e.g., when to check in source code, how to build and name a release). These rules then are disseminated across the involved development sites.

In order to build a single, virtual team the effectiveness of communication across development sites should be high. Two possible approaches to improve effectiveness are discussed below:

- First, as a preventive measure, it is possible to reduce the amount of communication across development sites by aligning the organizational structure with the software architecture (Conway, 1968). This requires that the architecture is fixed
4.3. Challenges in Global Software Development

(or, sufficiently fixed) to distribute the work and have development sites operate in parallel. In §4.3.4 we will further delve into this subject.

- Second, the use of collaboration tooling can help to create a single, shared environment across development sites. A single, shared environment can improve the efficiency of the communication between different groups that is anyhow needed, despite possible communication reduction efforts. Such an environment should form the entry point to revert to when communication across development teams is required. The environment should provide an overview of ‘who is who’ in the teams at the different development sites. At a minimum, contact information, role and responsibilities, and a mugshot (Herbsleb et al., 2005) should be included in the overview. In addition, these tools need to provide awareness of the availability of practitioners at other development sites. Collaboration technologies such as instant messaging, a wiki, or message-boards can lower the threshold of actually establishing contact and communicating with another site. The most appropriate tooling for a given situation depends foremost on the time difference that exists between development sites. The tools should allow architects and other practitioners to record design decisions (Herbsleb and Grinter, 1999a) in an easy and accessible way.

One word of warning, though. Although the use of collaboration tooling adds value, they are not a panacea. Face-to-face communication (ad hoc) remains essential for successful GSD (Herbsleb and Grinter, 1999a).

Architectural rules can help to overcome the challenge of building a team. First of all, a list of all employees involved in the project can be provided, along with their responsibility towards parts of the architecture. Second, the architectural rules can provide conventions on the use of collaboration tooling, by addressing what information is stored, where it can be found, and who is responsible for keeping the information up to date.

*No collective ownership*

When GSD organizations have dedicated owners of parts of the source code, this introduces delays in corrective maintenance. When it is not possible for practitioners to modify the source code that is not owned by them, additional (formal) communication with the owner becomes necessary to acquire a common understanding. In addition, it becomes necessary that the practitioner is able to transfer the sense of urgency he has to the owner of the source code.

The problems described above can be addressed by introducing collective ownership of (parts of) the source code and other documents. Collective ownership of source code and other documents means that all individuals collaborating in a software project (possibly located at multiple development sites) can work on any model or artifact that is accessible (Ambler and Jeffries, 2002). Often, collective ownership is supported by having one shared configuration management system with a single source code tree that is accessible (Herbsleb et al., 2005). As such, collective ownership reduces the view of ‘them against us’ often experienced in multiple teams (Herbsleb and Mockus, 2003). For large software development projects this could lead to collective ownership within
subteams (consisting of employees of all development sites) that work on a designated part of the system.

The possibility of changing source code freely should be accompanied by communication guidelines, a sound test set, and frequent builds of the software. Communication guidelines at a minimum should involve contacting other known users or developers of the source code and informing them that a change is being made. The test set ensures that a change does not introduce bugs in other parts of the source code or introduces non-compliance with architectural rules. Frequent builds address the possible delay between carrying out a change and verifying the correct behavior of the resulting source code (Herbsleb et al., 2005). Aforementioned topics need to be communicated uniformly across the GSD organization. Architectural rules can help to do that by defining the mandatory policy on these topics.

### 4.3.4 Challenge 4: Work distribution

In global software development, different teams from across the globe need to deliver working software in time. In addition to the communication challenges experienced by these teams (see Challenge 2), the organization of the teams itself and the variety of software development activities they need to perform play a significant role. We identified the following two issues in the literature:

6. Difficulty to align tasks and duties

7. No uniform process

**Difficulty to align tasks and duties**

The software architecture serves as a basis for distributing work. Ambiguities in the software architecture or frequent replanning of tasks leads to a lack of insight into the interdependence of tasks of the teams involved (Mullick et al., 2006). In addition, the distribution of work is hampered when engineering tasks are not understood. This all results in long discussions that add up to the delays already experienced in GSD (Herbsleb and Mockus, 2003). Furthermore, Herbsleb and Mockus show that the amount of discussions generally does not decrease in the course of the project.

A possible solution to this lack of understanding of tasks is to distribute tasks and work only when the architecture is stable enough. Architectural rules could specify when such is the case. For example, architectural rules could describe that the architecture description should elucidate how all ‘must-have’ requirements are addressed and that an independent verification of the architecture (e.g., by the quality assurance team) should have taken place.

**No uniform process**

As Mullick et al. show, it is essential to have a uniform strategy towards software engineering processes and team processes. A development organization that has different strategies towards these processes experiences delays (Mullick et al., 2006). An example of different strategies towards software engineering processes can occur during integration. Different approaches towards this integration phase exist: 1) a dedicated
integration team accepts all subsystems and is responsible for the integration, or 2) integration is the responsibility of all teams involved in delivering the subsystems that need to be integrated. When an organization chooses to have a dedicated integration team that integrates the subsystems, the integration team needs to assure itself of the correct working of the subsystems supplied. Furthermore, when issues arise, members of the development teams should stand by to provide assistance. Although this solution is considered to be less resource-consuming, it is hampered by time differences and causes delays because practitioners located at different sites communicate less effectively (Herbsleb and Mockus, 2003). Especially at integration time, the need is high for fast-paced interactions to quickly solve bugs once they are discovered (Herbsleb et al., 2005). Consequently, it can prove beneficial to have a dedicated team consisting of representatives from the different development teams at a single location integrate the subsystems into the desired release.

The example above shows that it is of paramount importance to have a uniform description of how integration should occur in projects. A non-uniform description of the integration process leads to delays because of ambiguity in task description. Similarly, process guidelines should exist for other software development processes, such as build processes and change management processes. Mullick et al., for instance, make note of broken builds because of a lack of a uniform configuration management strategy (Mullick et al., 2006). Examples on this topic are also given by (Herbsleb and Grinter, 1999a), reporting on problems when not having single-branched distributed change management across development sites.

Processes are an effective means to organize teams and their interrelationships and to distribute and communicate work that needs to be done (Herbsleb and Grinter, 1999a). Organizations need to put emphasis on team organization and communication processes in addition to software engineering processes. The team organization and communication processes need to focus on a communication structure (including a description of the roles and responsibilities of the various teams) and interactions between various types of teams (including the knowledge they exchange, at what stages or milestones in the development process they interact, and the frequency of interaction (Mullick et al., 2006)). Especially, the tasks and role of the architecture team should be exemplified so that all practitioners understand this pivotal role. Although these processes are not specifically aimed at catering for unplanned and informal communication, they can provide insight into the responsibilities of teams and the communication strategy across teams.

In conclusion, organizations involved in GSD need to focus on a set of processes that aids team organization and work distribution. These processes can have a technical topic, such as integration, branching, and releasing, as well as topics like work distribution and communication structure. Having these processes for software engineering and team organization and communication alone is not enough, though. In addition, it is necessary to communicate the processes consistently (Mullick et al., 2006) and to ensure that the processes are understood and followed.
4. Reassessing the Need for Practices for Architectural Compliance

4.4 Organizations’ Solutions to GSD Issues

Using the list of issues presented in §4.3 we studied two organizations involved in global software development. The aim of this study is to see how these organizations use architecture and architectural rules to overcome the issues. We conducted semi-structured interviews with two architects at each organization to obtain responses to the issues. In addition, we used the information collected in the case study performed at Organization A on which reported in Chapter 3. We verified our interpretation of the answers with the interviewees. For each organization, we first give a general impression of that organization. Next, we delve into the seven issues and show how the organization addresses each issue.

4.4.1 Organization A

Organization A uses four sites located on separate continents to develop software for different consumer electronics products. The development of each product is done in a project, in which a number of subsystems need to be integrated. Each of these subsystems is developed by a subsystem team located at a single development site. The subsystem team owns the source code of that subsystem. Subsystem teams consist of a subsystem architect, a configuration manager, an event manager, and several software engineers. Integration of the subsystems into the final product is done by a dedicated team located at the main development site. In total, about 300 employees work in the various teams.

The organization uses a product-line architecture to support various projects in which the consumer electronics products are developed. The architecture is maintained by a central architecture team located at a single development site. The architecture team maintains the architecture, by a.o. describing architectural rules in small, text-based documents. These rules only cover issues that hold for all subsystems. Issues pertaining to individual subsystems need to be addressed by subsystem architects. This results in a certain degree of freedom for subsystem teams.

1. Difficulty to initiate contact – As a general policy of Organization A, architects visit the largest remote development site one week each month. Other key individuals, such as senior software engineers and members of the integration team, do not travel that much. Practitioners indicated that this travel policy did not overcome the lack of trust as part of cultural challenges completely. Recently, Organization A was conducting a transfer of software development activities to the largest remote development site and retained the architecting activities at the main development site. This resulted in a lack of motivation of some of the employees at the remote development site because they felt this would decrease their influence on the major design decisions.

2. Difficulty to exchange information – Organization A uses a collaboration infrastructure that provides a detailed overview of the teams involved in a software development project, including the members, the roles, and responsibilities. In addition, the mugshots of the team members are made available as well. The collaboration infrastructure is linked with the configuration management infrastructure: information on builds,
4.4. Organizations’ Solutions to GSD Issues

releases, and problems is extracted from the configuration management system and published on each subsystem team’s website. Together with a description of the members of the team and the team organization, this website is one of the primary sources of information for other subsystem teams. Besides using the website to obtain information, the organization uses email communication and instant messaging technology regularly to allow discussions between the subsystem teams and between the architecture team and subsystem teams. As a matter of fact, Organization A regards team composition and team contact details as the most important information, as was shown in Chapter 3. Aforementioned collaboration infrastructure lowers the threshold for exchanging information at Organization A. However, other issues with GSD still burden fully effective exchange of information.

3. **Difference in sense of urgency** – The organization experiences difficulty in maintaining a shared sense of urgency, judging by an example given:

“(….) when programming errors or compiler warnings occurred, it was difficult to get practitioners from the involved development site to react on this warning. When this succeeded, practitioners from that development site did nearly everything to just remove the warning message, hardly paying attention to the actual semantics of the implemented solution. Practitioners from the main development site actually dug into the warning message, tried to understand the root cause of this message, and removed the root cause.”

Although the latter approach took much time, Organization A was much more confident that this approach, once used throughout the organization, would be more beneficial.

4. **Difficulty to build a team** – Organization A has a nearly exact mapping of the organizational structure to the architecture. For all subsystems defined in the architecture, a subsystem team exists. In addition, architectural rules are the responsibility of the architecture team. Some of the architectural rules of Organization A define allowed dependencies between subsystems; these dependencies closely resemble the communication processes in place. In conclusion, the architecture determines the team structure so that the definition of teams is fixed. This leads to increased formality in the communication between these teams.

5. **No collective ownership** – The configuration management infrastructure in place at Organization A supports distributed change management. Each subsystem team is owner of the source code of that subsystem. Consequently, the ownership of the source code is distributed. Communication across subsystem teams on source code topics is structured via change requests and problem reports, and formalized in architectural rules. The organization experiences this as highly formal and inadequate; the major issues are discussed using other channels than change requests. As a result, change requests undergo delays and do not always represent reality.

6. **Difficulty to align tasks and duties** – Alignment of tasks and responsibilities is mainly done via the architecture. Requirements are assigned to subsystems, which have dedicated resources. Therefore, it is clear what activities will be performed by what
subsystem teams, even by what practitioners within those teams. Because these tasks and responsibilities are defined up-front, it is necessary to conduct formal verifications to determine whether reality is in line with the rules. Nevertheless, the tasks performed by the subsystem teams are not reviewed regularly by the architecture team nor the quality assurance team.

7. **No uniform process** – When we observed the architectural rules at Organization A (see Chapter 3), we noticed that Organization A has a strong emphasis on architectural rules from a configuration management perspective: naming conventions, coding guidelines, and process guidelines on releasing, integrating, and deploying source code are regarded as important by the organization. These guidelines pertain to the use of a single configuration management system. In addition, we observed much emphasis on software engineering processes and communication infrastructure. What lacks are detailed guidelines on inter-team communication and collaboration and a sound mechanism to disseminate the most important architectural decisions. Especially when unplanned, social contact is hard, it is essential to establish and announce communication guidelines between teams to disseminate the essential (architectural) information, such as the major architectural decisions. This prevents subsystem teams from feeling put aside and trailing on reality, as was the case at Organization A.

Although Organization A has guidelines pertaining to the software engineering processes as described above, the organization often deviated from these guidelines. This results in e.g., ambiguity on the integration process; the guidelines describe that all subsystems were delivered to a dedicated integration team, responsible for the full integration to the final software product. Practice shows a more staged approach towards integration. Furthermore, the visibility of the architecture team is decreasing because of a lack of resources and the perception that the architecture team is trailing on projects. Verification of compliance with the architectural rules is not implemented at Organization A. As a result of this, multiple processes co-exist in practice.

### 4.4.2 Organization B

Organization B develops business administration systems for various customers. Software development occurs by one development team spread across two sites at different continents. Organization B does not have dedicated architects – rather, architecting is a joint effort by the complete development team. Since 2006, Organization B uses these two development sites. In total, about 100 employees work in projects at these two development sites.

Although Organization B acknowledges that the differences between its customers may result in different architectural solutions, experiences have led to a certain convergence on the architecture that can be used most often. The software architecture consists of four layers which are highly decoupled by using principles like inversion of control, see e.g., [Walls and Breidenbach, 2005], and is supported by an extensive toolset aiding the development approach of Organization B.

The development approach of Organization B uses several practices from the agile development domain ([Ambler and Jeffries, 2002](#)). Examples of these practices include
pair programming and test-driven development. Test-driven development serves as a good basis for performing integration activities \cite{Beck2002}. Integration is done continuously by running all tests automatically and providing a response to the team.

1. **Difficulty to initiate contact** – In setting up the activities at the remote development site, specific attention was paid to selecting and hiring employees with corresponding work ethics and attitude. Furthermore, the management and several key individuals of the main development site often travel to the remote site. This traveling results in getting to know the people at the other site and establishing a shared approach using exactly the same methodology and tooling for software engineering. Since starting with the remote development site, still every employee hired at the remote development site visits the main development site within two months. Finally, employees from the remote development site regularly visit the practitioners at the main development site to exchange ideas and have face-to-face discussions on project-specific matters.

2. **Difficulty to exchange information** – The work of Organization B is supported by a configuration management system and an integrated issue-tracking system. Furthermore, Organization B uses a wiki as a collaboration tool to capture discussions, provide documents, and relate information on configuration management and issue-tracking. The use of the wiki started as a shared initiative and now is an established practice. All important documents are stored on the wiki and all issues are reported uniformly. All software engineers are involved in daily stand-up meetings with video-conferencing facilities.

3. **Difference in sense of urgency** – Organization B is not really bothered by this issue. Frequent traveling across development sites (including informal activities) helps to established a shared sense of urgency. In addition, all employees spend time at the main development site to get to know the customer’s context. This enables the sites to relate communication efforts back to requirements of that customer.

4. **Difficulty to build a team** – First of all, the sites of Organization B regard each other fully as ‘peers’. This contributes to a shared team understanding. Furthermore, having highly communicative meetings across the development sites throughout the course of the project ensures that a single, uniform team exists across development sites.

5. **No collective ownership** – Organization B uses a single configuration management system in which all software engineers are allowed to view and modify all parts of the source code. Programming occurs in pairs that are often changed. In addition, no distinction between software engineering tasks is made between the development sites. Consequently, the organization has a high degree of collective ownership. In case a practitioner makes a change to a part of the source code that is relatively unknown, the practitioner contacts other software engineers on the change that is to be made. The correct effect of the change is verified by the different kinds of tests that run automatically.

6. **Difficulty to align tasks and duties** – Since no distinction between software engineering tasks is made at Organization B, all practitioners within a project team use a single task list for dividing the work. Given the nature of the projects and the high pressure put on them, Organization B uses a very light-weight approach for capturing architectural knowledge (including architectural design decisions) and making this
knowledge explicit. This knowledge is used to introduce new practitioners into the architecture and the architectural rules that apply.

7. **No uniform process** – Organization B uses SCRUM [Schwaber and Beedle, 2001] as a software development process. As mentioned at the issue of difficulty to exchange information, frequent communication between practitioners from the development sites results in common understanding of the development process. Furthermore, an organization-wide shared vision on software development gives support for following that process. Since a while, releases are made from both the main development site and the remote development site, pointing out the similarity in the processes. This is further exemplified by a quote of a software engineer of Organization B:

“It does not matter whether I work at the remote development site or at the main development site.”

4.5 Conclusions

This chapter reports on the use of architectural rules to overcome global software development issues. We have structured the challenges and solutions as identified in the literature. As a result, we have defined four challenges in GSD: time difference and geographical distance, culture, team communication and collaboration, and work distribution. We have also shown that these challenges cannot be fully regarded in isolation; they influence each other.

We observe that the solutions that are proposed by the literature pertain to both the product (i.e., the structure of the software) and the process. Furthermore, we have exemplified how the solutions can be implemented by using architectural rules or process measures.

We used the list of GSD issues when interviewing two organizations to identify what solutions the organizations use to overcome these issues. We learned that some of these solutions can be described by using principles and statements (i.e., architectural rules) on the software architecture that must be complied with throughout the organization.

Table 4.1 summarizes the results of our analysis. Organization A has a clear focus on software architecture throughout the organization. Mechanisms such as having an architecture team and architectural rules on different topics exist to remain in control of the architecture. The organization adheres to a waterfall approach in defining and disseminating architecture throughout the organization. With this type of approach, it is easier to check and assure that the architectural rules are complied with. In addition, it is possible to e.g., measure to what extent architectural rules help in GSD. Nevertheless, this type of approach also requires to verify backward compliance. However, this verification does not take place at Organization A. Organization B has a more agile approach towards architecture. Although the organization has several architectural rules in place, the organization uses a number of practices and measures in the development process as solutions to overcome the GSD issues.
### 4.5 Conclusions

Table 4.1: Overview of organizations’ solutions to GSD issues. Solutions in the form of architectural rules are marked with [R]. Process measures are marked [C].

<table>
<thead>
<tr>
<th>Issue</th>
<th>Organization A</th>
<th>Organization B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Difficulty to initiate contact</td>
<td>Many trips of architects to software engineers [C]</td>
<td>Periodic traveling of managers and key individuals [C]</td>
</tr>
<tr>
<td></td>
<td>Not all key individuals travel [C]</td>
<td>Developers travel when hired [C]</td>
</tr>
<tr>
<td>2. Difficulty to exchange information</td>
<td>‘Yellow-pages’, subsystem website [R] [C]</td>
<td>Use of a wiki [C]</td>
</tr>
<tr>
<td></td>
<td>Frequent highly communicative meetings [C]</td>
<td></td>
</tr>
<tr>
<td>3. Difference in sense of urgency</td>
<td>Local bug fixing [C]</td>
<td>Shared view of customer context across development sites [C]</td>
</tr>
<tr>
<td>4. Difficulty to build a team</td>
<td>Dedicated subsystem teams, communication through architecture [R]</td>
<td>Frequent highly communicative meetings [C]</td>
</tr>
<tr>
<td>5. No collective ownership</td>
<td>Distributed code responsibility/ownership [R] [C]</td>
<td>Shared code responsibility/ownership [C]</td>
</tr>
<tr>
<td></td>
<td>Frequent builds [C]</td>
<td></td>
</tr>
<tr>
<td>6. Difficulty to align tasks and sites</td>
<td>Alignment via architecture [R]</td>
<td>Joint planning [C]</td>
</tr>
<tr>
<td></td>
<td>Formal compliance-checking needed [C]</td>
<td>Frequent communication [C]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Informal, continuous compliance-checking [C]</td>
</tr>
<tr>
<td>7. No uniform process</td>
<td>CM-tooling is present [R]</td>
<td>CM-tooling is present [R]</td>
</tr>
<tr>
<td></td>
<td>Clear “official” process, multiple “real” processes in practice [C]</td>
<td>SCRUM process with test-driven development [C]</td>
</tr>
</tbody>
</table>
Architectural rules prove valuable in handling some of the GSD issues in GSD. Organization A mainly uses architectural rules that pertain to the product as well as some additional process measures. Sometimes, architectural rules pertaining to the product are meant to induce the necessary processes (issues 4 and 6 in Table 4.1). However, we also observe that architectural rules pertaining to the product may induce difficulties for addressing GSD issues for Organization A, such as maintaining a uniform process. Organization B does not so much focus on architectural rules but rather reverts to measures in the development process to tackle the issues related to GSD and ensure compliance with architectural rules. Organization A, on the other hand, leaves the teams a certain degree of freedom in these processes and does not verify compliance with these processes nor the architectural rules.

Our study shows that architectural rules are not a solution for all GSD issues identified: especially cultural challenges and team collaboration challenges are not addressed by using architectural rules in the organizations.

Further, we have to take into account the differences between the two organizations, such as differences in complexity and size (Perry and Kaiser, 1991). Nevertheless, we feel that a number of light-weight practices in place at Organization B can be transferred successfully to Organization A. For example, two-way traveling greatly helps in building a team and, consequently, in having development sites regard each other fully as ‘peers’. In addition, to build a collective sense of urgency, the proportion of architect’s work and developer’s work (regardless of what development site is their stand) could be distributed more uniformly across development sites. The other way round, Organization B may need to place more emphasis on architectural rules in addition to process measures when development scales up. We conclude that rules regulating a combination of both the architectural rules and measures pertaining to the architecture process proves valuable in handling GSD challenges.
Product Practices for Architectural Knowledge Management in Global Software Development

Chapters 3 and 4 have provided insight into how compliance with architectural rules can be secured in global software development organizations. The main findings obtained in this research indicate that process measures seem prevalent over measures that can be taken in the product. In this study, we determine whether practices related to the product can help in capturing architectural knowledge and serve as measures to increase the quality of the software product.

5.1 Introduction

In the discipline of software architecture, increasing attention is paid to capturing architectural decisions and underlying rationale as important means to describe software architectures. In this way, architectural knowledge serves as an enabler for high quality software products: an increased understanding of the ‘why’ behind certain decisions helps to reduce the implementation and maintenance effort of the resulting system (Bosch, 2004; Lago et al., 2010).

The increasing trend towards global software development (Aspray et al., 2006) poses an additional burden towards developing high quality software products, since additional challenges come into play. These challenges include temporal, geographical, and socio-cultural challenges (Holmström et al., 2006).

It has generally been acknowledged that measures in the realm of communication and coordination strategies can address the challenges involved in GSD (Agerfalk et al., 2005; Clerc et al., 2007b). These measures include specific practices to share architectural knowledge in a global context, see e.g., Chapter 3. Aside from these measures that focus on the process, measures can be taken in the software product as well; taken measures can serve as a proxy for the quality of the software product. However, we are currently lacking insight into what product-based architectural knowledge measures exactly contribute to the quality of the software product.
5.2 Related Work

Recently, a growing trend towards capturing and managing architectural knowledge to represent the architecture of a software system can be observed in the literature (Bosch, 2004; Avgeriou et al., 2007; Kruchten et al., 2006; Babar et al., 2007). Even more recently, work has been performed to define what architectural knowledge exactly entails (de Boer and Farenholtz, 2008). When we regard architectural knowledge in the realm of global software development, we observe contrasting views on the amount of knowledge to be made explicit in documentation, e.g., in (Ågerfalk and Fitzgerald, 2006; Ramesh et al., 2006). This contrast is further exemplified by our previous work (Clerc et al., 2007b; Lago et al., 2010). The work reported so far primarily seems to focus on measures related to the development process. Bass et al. (2007a) have identified a best practice template to describe these process measures. In Chapter 4 we compared two organizations involved in GSD, and found that one organization was very successful by primarily applying process measures.

Existing literature does not so much include measures that can be taken in the software product\(^1\). For identification of measures present in software products, one may use an auditor’s approach using an explicit frame of reference for an audit, such as the one described in (ISO/IEC, 2000).

\(^1\)A software product is defined as a “set of computer programs, procedures, and possibly associated documentation and data” (ISO/IEC, 2000). For this study, we focus on (possible intermediate) tailor-made solutions, since we do not include commercial-off-the-shelf components in this study.
5.3 Context and Approach

In this section, we provide the context for this research by describing the tool used by the IT consulting firm to perform software product assessments. Furthermore, we describe the research approach.

5.3.1 Software quality evaluation framework

We have performed research at a Dutch IT consultancy firm, Organization C, that regularly performs independent software product assessments for its clients. These clients are e.g., banks, IT departments of ministries, and non-governmental organizations. Organization C has laid down its experience in performing over 20 of these assessments in a reusable framework containing evaluation criteria (i.e., a software quality evaluation framework). These evaluation criteria are backed by literature as to positively impact one or more quality attributes (as e.g., specified in ISO/IEC [2001]) and hence serve as measures that can be taken in the software product to increase its quality.

The software quality evaluation framework contains 93 evaluation criteria. Each criterion is described by its name, a description, a motivation, an assessment method for the criterion, and inferred consequences of applying the criterion. In addition, the framework defines the scope of the criterion (e.g., “does the criterion apply to the software product as a whole, or to subsets, such as documentation or source code?”). Furthermore, the framework relates each evaluation criterion to certain quality attributes. We provide an abstracted example of a evaluation criterion used by Organization C below:

<table>
<thead>
<tr>
<th>Name</th>
<th>No code duplication.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Software code should not contain unnecessary duplication.</td>
</tr>
<tr>
<td>Motivation</td>
<td>Code duplication increases the change effort of the software product; changes need to be made at multiple locations in the source code.</td>
</tr>
<tr>
<td>Evaluation method</td>
<td>Use automatic code analyzers to detect code duplication.</td>
</tr>
<tr>
<td>Consequences</td>
<td>Removing code duplication may lead to refactoring.</td>
</tr>
<tr>
<td>Quality attributes affected</td>
<td>Analysability, Changeability, Stability, Testability.</td>
</tr>
</tbody>
</table>

5.3.2 Research approach

Since we are primarily interested in how architectural knowledge can help to improve the quality of software products, we focus on the evaluation criteria that specifically deal with the topic of architectural knowledge. We define key elements of architectural knowledge by researching definitions of architectural knowledge posed by relevant literature (see §5.3.3). Using these key elements of architectural knowledge, we study the set of evaluation criteria in the software quality evaluation framework and select those criteria that, when applied, make explicit one or more key elements of architectural knowledge (see §5.3.4).

Using the subset of the resulting so-called architectural knowledge evaluation criteria, we analyze eight pre-selected software product assessment reports to determine
what key elements of architectural knowledge are applied in the products. These assessments concern four products that have been developed using GSD and four products that have been developed on a single site. The result of this analysis is presented in §5.4.

5.3.3 Key elements of architectural knowledge

To identify what key elements constitute architectural knowledge, we performed a thorough analysis of definitions listed in relevant literature. We identified relevant literature by focusing on the reports produced in the SHARK workshop and the WICSA conference series, both since 2006. We searched these articles for definitions of architectural knowledge. Furthermore, we included recent work that describes a systematic review on how architectural knowledge is defined and how different definitions are related (de Boer and Farenhorst, 2008). In their paper, De Boer and Farenhorst list definitions of architectural knowledge from fourteen papers, collected after synthesis of a total set of 751 found initially. While analyzing all collected definitions, we identified similar parts in these definitions. These similar parts included synonyms like “architectural decisions” and “architectural design decisions”, or “design” and “architectural design”. We mapped the synonyms identified onto a single key element of architectural knowledge. The resulting list of key elements of architectural knowledge is shown in Table 5.1.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Des</td>
<td>The architectural design itself</td>
</tr>
<tr>
<td>Ass</td>
<td>Assumptions that were made during the architectural design and underpinning design decisions</td>
</tr>
<tr>
<td>Env</td>
<td>Linkage to the environment</td>
</tr>
<tr>
<td>Dec</td>
<td>Design decisions</td>
</tr>
<tr>
<td>Dep</td>
<td>Interdependencies between the design decisions</td>
</tr>
<tr>
<td>Map</td>
<td>Mapping of design decisions to requirements, needs, constraints, design, and implementation</td>
</tr>
<tr>
<td>Dom</td>
<td>The domain analysis</td>
</tr>
<tr>
<td>Pat</td>
<td>Architectural patterns used</td>
</tr>
<tr>
<td>Alt</td>
<td>Design alternatives evaluated</td>
</tr>
<tr>
<td>Rat</td>
<td>Rationale</td>
</tr>
</tbody>
</table>

5.3.4 Architectural knowledge evaluation criteria

Following the approach described in §5.3.2, we selected those evaluation criteria from the software quality evaluation framework that concern architectural knowledge. This resulted in fourteen out of 93 evaluation criteria to be selected. Each of these fourteen evaluation criteria make explicit at least one key element of architectural knowledge.
5.4 Results

For reasons of brevity and confidentiality we do not provide an exhaustive description of the evaluation criteria. Rather, we list the names of the selected architectural knowledge evaluation criteria in Table 5.2.

Table 5.2: Architectural knowledge evaluation criteria

<table>
<thead>
<tr>
<th>ID</th>
<th>Criterion Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crit. 1</td>
<td>Appropriate views and viewpoints</td>
</tr>
<tr>
<td>Crit. 2</td>
<td>Bottlenecks identified</td>
</tr>
<tr>
<td>Crit. 3</td>
<td>Build and deployment documentation</td>
</tr>
<tr>
<td>Crit. 4</td>
<td>Decomposition and layering</td>
</tr>
<tr>
<td>Crit. 5</td>
<td>Design patterns and architectural patterns</td>
</tr>
<tr>
<td>Crit. 6</td>
<td>Design rationale</td>
</tr>
<tr>
<td>Crit. 7</td>
<td>Document application architecture and design</td>
</tr>
<tr>
<td>Crit. 8</td>
<td>Explicit external dependencies</td>
</tr>
<tr>
<td>Crit. 9</td>
<td>Explicit internal dependencies</td>
</tr>
<tr>
<td>Crit. 10</td>
<td>Fail-over</td>
</tr>
<tr>
<td>Crit. 11</td>
<td>Load balancing</td>
</tr>
<tr>
<td>Crit. 12</td>
<td>Rationale</td>
</tr>
<tr>
<td>Crit. 13</td>
<td>Scalable capacity</td>
</tr>
<tr>
<td>Crit. 14</td>
<td>Single-points-of-failure</td>
</tr>
</tbody>
</table>

The selected evaluation criteria focus on either building blocks of architecture (e.g., Des, Env, and Dom in Table 5.1) or on the decision-making process and its results (e.g., Ass, Dec, Dep, Dom, Pat, Alt, Rat). In addition, we observe that the quality attributes affected most by the selected architectural knowledge evaluation criteria are ‘analysability’, ‘changeability’, and ‘manageability’. Each of these quality attributes are positively impacted by nine of the fourteen selected architectural knowledge evaluation criteria.

5.4 Results

This section lists the results of our research. First, we provide a brief characterization of the context in which the selected software products were developed in §5.4.1. Next, we provide the results of our evaluation in §5.4.2 by providing a scale and contrasting the products that were developed using global software development with products that were developed using collocated development.

5.4.1 Characterization of the selected software development projects

We have included eight software development projects as a basis for our study. These projects focused on the development of business administration systems and not on e.g., commercial-off-the-shelf products. The evaluations of these projects were performed in the period late 2004 until late 2007. Four of these projects took place using multiple,
globally distributed development sites. The remaining four were developed either at
the customer’s location or at a single location of the IT service provider developing the
product close to the customer. The global software development projects included 12
project members on average (range 3–23); the others included 5 project members on
average (range 3–12). Programming languages and technologies in use included .Net
(C#), Java, Oracle, and various workflow engines.

5.4.2 Evaluation results

We studied the assessment reports of the eight selected software product assessments
to determine to what extent the evaluation criteria are complied with in the software
products. Since the assessment reports did not provide us with a single, uniform scale,
we devised an ordinal scale to identify the degree of implementation, see Table 5.3. This
scale does not only focus on whether a criterion has been applied, but also on the level
of description of the application, since that information is key to transferring knowledge
to other project members – especially in the context of global software development.
Consequently, this scale serves as realignment of the auditors’ judgment over time.
Regarding the scale: several reasons may exist for rating the implementation of an
evaluation criterion with ‘+'. For example, the rate ‘+' is given if either the description
of the application of the criterion is incomplete, or if the description is unclear in the
perception of the auditors.

Table 5.3: Scale for evaluation criteria implementation

<table>
<thead>
<tr>
<th>Scale</th>
<th>Description</th>
<th>Motivation</th>
</tr>
</thead>
<tbody>
<tr>
<td>––</td>
<td>Not applied</td>
<td>The criterion has not been applied in the software product</td>
</tr>
<tr>
<td>–</td>
<td>Applied only</td>
<td>The criterion has been applied, but not described</td>
</tr>
<tr>
<td>+</td>
<td>Applied moderately explicit</td>
<td>The criterion has been applied, some information on its application is described</td>
</tr>
<tr>
<td>++</td>
<td>Fully explicit</td>
<td>The criterion has been applied and its application is described in full detail</td>
</tr>
</tbody>
</table>

Table 5.4 lists the results of the software product assessments that have been evaluated.
We grouped the products into global and non-global products, depending on whether
the product has been developed using GSD or not. Furthermore, we applied a further
characterization of the architectural knowledge evaluation criteria from Table 5.2 using
the key elements of architectural knowledge. We identified four clusters of architec-
tural knowledge evaluation criteria: documentation, architectural knowledge (including
the key elements Ass, Alt, Dec, Dep, and Rat), solution fragment (including the key
elements Des, Dom, Pat, and Env), and performance (including the key element Map).
Elements labelled “n/a” in Table 5.4 indicate that the selected evaluation criterion was
not inside the scope of that specific software product assessment (e.g., it was deemed
5.4. Results

Table 5.4: Ordered results of assessment analysis

<table>
<thead>
<tr>
<th>Category</th>
<th>Criteria</th>
<th>A</th>
<th>B</th>
<th>D</th>
<th>H</th>
<th>C</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>global</td>
<td></td>
<td></td>
<td></td>
<td>non-global</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td>Crit. 1</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Crit. 3</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>--</td>
<td>n/a</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 7</td>
<td>--</td>
<td>+</td>
<td>+</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td>Rationale</td>
<td>Crit. 12</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 6</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Solution Fragment</td>
<td>Crit. 4</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Crit. 5</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>--</td>
<td>n/a</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 8</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 9</td>
<td>n/a</td>
<td>n/a</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 10</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>--</td>
</tr>
<tr>
<td>Performance</td>
<td>Crit. 2</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Crit. 11</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>Crit. 13</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Crit. 14</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>--</td>
</tr>
</tbody>
</table>

not applicable for some reason.

We make the following observations on the application and description of the architectural knowledge evaluation criteria in the software product assessments under study. We list our observations per cluster of architectural knowledge evaluation criteria:

Documentation – Crit. 7 – Documenting application architecture and design is applied in three software products that have been developed using GSD. Furthermore, in two of these products, its application has been described as well. Yet, we observe that Crit 1. – Appropriate views and viewpoints has either not been applied in the architectural descriptions of the products to address the needs of the stakeholders involved (IEEE, 2000) (Product H), or has not been included in the assessment scope (products A, B, and D).

We observe that Product G explicitly incorporates views and viewpoints whereas Product E has used views and viewpoints without a sufficient description or explanation of its application. Furthermore, the architecture description of Product G elaborates on each view in additional reference documents.

Rationale – The two evaluation criteria (Crit. 6 – Design rationale and Crit. 12 – Rationale) that fall in this category are not always applied in the software products under study; when they are applied (at products A, D, C, F, and G), its application has not been described. E.g. in the evaluation report of Product A, developed in a large GSD project, the auditors of this assessment conclude that:

“(…) the description of the rationale and assumptions in the architecture description is very minimal.”
5. Product Practices for AKM in GSD

Solution Fragment – Crit. 4 – Decomposition and layering and the use of Crit. 5 – Design patterns and architectural patterns in general are well-known approaches (measures) to reduce complexity in a software product. The application of these measures, however, is rather scarce, both in products developed using a single development site and using GSD. Apart from that, the only explicit occurrence of a measure in this category is with Crit. 8 – Explicit external dependencies at Product B. Further examination of the assessment of Product B showed no specific circumstances in which this measure is applied. It is applied and described in a functional design and in the installation manual of the product. In Product A, on the other hand, the criterion is not applied, causing the product to be assessed as ‘overly complex’.

Performance – Various architectural knowledge evaluation criteria have to do with a specific quality criterion, i.e., performance (termed ‘time behavior’ in [ISO/IEC, 2001]). We observe that in most cases, these architectural knowledge evaluation criteria either have not been applied (at most products developed using GSD) or have been left outside the scope of the assessment (at most products developed using a single development site).

5.5 Discussion of the Results

In this research, we showed what key elements constitute architectural knowledge. Using these key elements and a series of software product assessments, we determined which key elements of architectural knowledge are made explicit in the software product (so-called architectural knowledge evaluation criteria). We focused on differences between products that were developed using global software development and those that were developed using a single development site. As such, this research sheds light on what architectural knowledge measures are applied in GSD to improve the quality of software products.

If a similar approach was used during development of the software product instead of only when development has been completed, feedback could be given to the software development organizations in a timely fashion. This would allow for adequate measures to be taken to increase the quality of the software product during the development phase.

In contrasting software products developed using GSD with products developed on a single development site, we did not find significant differences between the two groups. We do observe some overall results on the application of architectural knowledge in products developed using GSD.

Software products developed using GSD, as those developed using a single development site, typically have an architectural description. This confirms an increased attention towards documentation in GSD, as expressed in [Agerfalk and Fitzgerald, 2006], although we have not found evidence for an overly focus on documentation in these products. On the other hand, the architectural descriptions generally do not seem to be organized using views and viewpoints to address specific stakeholder concerns [Clements et al., 2003]. Apparently, the benefits of a view-based architectural description as described by e.g., [Sangwan et al., 2006] are not yet widely acknowledged.
Benefits of capturing architectural knowledge that enable architecting as a decision-making process (i.e., assumptions, alternatives, and rationale) are not yet widely accepted in GSD. This is in line with an earlier study we conducted (see Chapter 2) but still shows that the perceived benefits need to be communicated further, especially in GSD environments.

Architectural knowledge that focuses on solution fragments is not regularly made explicit in software products or their documentation. This poses a question on how knowledge on these topics is shared in the GSD projects. These findings appear to contradict the focus on overly (solution-based) documentation generally propagated by GSD literature.

Performance is an important quality attribute which, according to the auditors, and acknowledged by [Bachmann and Bass, 2001], requires to be addressed by architectural knowledge. It is striking that in hardly any software product information on bottlenecks, scalability, load balancing, or single-points-of-failure can be found. Furthermore, we learned that this cluster of architectural knowledge seems to be not applicable in projects running on a single site; surely, performance-dependent systems are not confined to being developed using GSD.

Several architectural knowledge evaluation criteria were left outside of the scope of the software product assessments we studied. Yet, the criteria used in this study are universal criteria for high-quality software products. Consequently, the understanding of the importance of these evaluation criteria and their benefits still can be improved by explicitly incorporating these criteria in an evaluation framework.

5.5.1 Limitations

Our study was performed on a subset of software product assessments on business administration systems performed by an IT consultancy firm in The Netherlands. We use this study to gain insight into measures taken in software products to make explicit architectural knowledge. As such, this study does not aim to give a broad overview of the status quo of software product quality, but aims at sparking a discussion on the use of product-based measures related to architectural knowledge to increase software product quality in general and, more specifically, in GSD.
Identifying Practices for Architectural Knowledge Management in Global Software Development

In the previous two chapters, we have described studies that were focused on identifying how architectural knowledge management occurs in global software development. The results showed that process practices can provide solutions to issues involved in GSD. Furthermore, we found that product practices for architectural knowledge management do not appear to contribute substantially to the quality of products developed using GSD. In this chapter, we take a different perspective. We identify whether architectural knowledge management can leverage the solutions that are available from the requirements engineering discipline to address the challenges in GSD.

6.1 Introduction

In the past few years, an increasing interest in architectural knowledge management can be recognized in the software architecture community (Jansen and Bosch, 2005; Kruchten et al., 2006; Babar et al., 2007; van der Ven et al., 2006b). Generally, architectural knowledge is regarded as important to guide the development and evolution of software systems (Kruchten et al., 2006).

With the trend of global software development, architectural knowledge management becomes even more important due to challenges that arise as a result of the geographical, temporal, and socio-cultural distance innate to GSD (Holmström et al., 2006). Overviews of the challenges in GSD have been widely reported in (Battin et al., 2001; Ågerfalk et al., 2005; Holmström et al., 2006; Clerc et al., 2007a,b) and include lack of informal contact, language differences, and coordination complexity, see e.g., (Ågerfalk et al., 2005).

Solutions to overcome GSD challenges generally deal with the way individuals interact with each other in a distributed setting. These solutions are provided in terms of communication and coordination strategies (Herbsleb and Griner, 1999a; Herbsleb, 2007; Lanubile et al., 2003). In these strategies, recording decisions about the architec-
6. Identifying Practices for AKM in GSD

ture plays a pivotal role (Clerc et al., 2007a; Herbsleb and Grinter, 1999a). However, we are currently lacking detailed insight into architectural knowledge management practices that can effectively be applied in a GSD setting, following observations from the literature on challenges in knowledge sharing in a GSD situation (Desouza et al., 2006; Balaji and Ahuja, 2005).

To address this lack of insight, we build on the discipline of requirements engineering. The requirements engineering discipline is a well-discussed example of a discipline that becomes challenging in GSD, see e.g., (Hsieh, 2006; Damian, 2007). We observe a close resemblance between a set of requirements for a software system and a set of architectural decisions taken for that software system: what one person regards as requirements for a software system, another person may regard as architectural decisions (van Vliet, 2008; de Boer and van Vliet, 2009). Furthermore, we conjecture that the same challenges that the requirements engineering discipline faces with GSD hold for architectural knowledge management practices in GSD as well: as with requirements, architectural decisions too need to be communicated across different sites in order to maintain a shared vision of the software system that is designed.

We have constructed a set of architectural knowledge management practices based on a study of relevant literature on the requirements engineering discipline related to GSD. We describe these practices using a light-weight pattern language. The elements of this pattern language and its application on describing architectural knowledge management practices may help organizations in making a well-supported choice between architectural knowledge management practices for application in GSD projects. We provide an initial validation of the usefulness and effectiveness of these practices through semi-structured interviews on these practices within a large IT service provider.

This chapter is structured as follows. In §6.2 we list the approach we applied for this research. Next, in §6.3 we provide an overview of the resulting architectural knowledge management practices in GSD. We then provide an initial validation in §6.4 and list our conclusions in §6.5.

6.2 Research Approach

In this section, we describe our approach to select relevant literature for this research in §6.2.1 and describe a light-weight pattern language for the description of the architectural knowledge management practices in §6.2.2.

6.2.1 Literature research

We built a representative subset of relevant literature on the topic of requirements engineering in relation to global software development. First, we identified important software engineering conferences on this subject, such as the ICSE GSD workshops, the ICGSE ’06 and ’07 conferences, and the RE conferences. Subsequently, we collected the proceedings of these conferences. In addition, we selected special issues from relevant journals (Communications of the ACM, IEEE Software).
6.2. Research Approach

In our search to collect GSD requirements engineering practices that can help in overcoming GSD challenges, we scanned the abstract, introduction, and conclusion of all contributions. Contributions which explicitly reported on validated practices were studied in full detail.

After having collected the GSD requirements engineering practices, we translated these practices, if needed, to the discipline of architectural knowledge management. When the requirements engineering practices mentioned “requirements”, we translated this into “architectural decisions”; when the practices mentioned the “requirements engineering discipline”, we translated this into the “architecting phase” or “architecture development phase”, following the similarities discussed in §6.1. In this way, we ensure that we stay fully in line with the practices as they were initially reported in the requirements engineering literature; we list the references to the requirements engineering literature throughout the description of the practices.

6.2.2 A pattern language for describing architectural knowledge management practices

In parallel with the literature study, we devised a light-weight pattern language to structure and describe the potential GSD architectural knowledge management solutions posed by the literature. We used the work of (Bass et al., 2007; Tyree and Akerman, 2005), based on (Gamma et al., 1994) as a basis to define a light-weight pattern language. We added relevant insights from the field of architectural knowledge management, such as the architectural knowledge management strategy organizations may adhere to (Babar et al., 2007; Hansen et al., 1999).

Table 6.1 lists the results. The elements of this pattern language and its application on describing architectural knowledge management practices may help organizations in making a well-supported choice between architectural knowledge management practices for application in GSD projects.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Practice Name</td>
<td>An unique name describing the practice</td>
</tr>
<tr>
<td>Intent</td>
<td>The goal for which this practice can be applied</td>
</tr>
<tr>
<td>Motivation</td>
<td>An example in which or discussion why the given practice proves to be useful</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>Prerequisites or limitations that exist when applying the practice</td>
</tr>
<tr>
<td>Benefits</td>
<td>The benefits, side effects and trade-offs</td>
</tr>
<tr>
<td>Drawbacks</td>
<td>The drawbacks or negative effects of the practice</td>
</tr>
<tr>
<td>Strategy</td>
<td>The knowledge management strategy which is supported best by the practice, e.g., codification or personalization (Hansen et al., 1999)</td>
</tr>
</tbody>
</table>
6.3 Architectural Knowledge Management Practices for GSD

This section lists the practices for architectural knowledge management to overcome GSD challenges, distilled from the literature on the requirements engineering discipline.

When a practice does not refer to any requirements-specific terminology or process, we have not applied any modifications. For a practice that does, we translate the given requirements engineering-specific element to an appropriate counterpart in the realm of architectural knowledge management (see §6.2.1), following the insights from e.g., (van Vliet, 2008). We provide an overview of the changes we made.

6.3.1 Frequent interaction across sites

Practice Name – Frequent interaction across sites (Damian and Zowghi, 2002).

Intent – This practice intends to let practitioners from different development sites interact frequently with each other.

Motivation – In knowledge-intensive tasks such as requirements engineering and architecting, the geographical distance and possible time difference burden effective sharing of architectural knowledge. This may result in language and terminology problems. More frequent interaction across sites helps to address these problems. In addition, more frequent interaction helps in building trust across sites, as shown in Chapter 4. Furthermore, it helps in gaining awareness of the local working context of architects, developers, and other representatives which eases understanding across sites.

Frequent interaction can be implemented in a variety of ways: organizations may use on-site management visits (Bass et al., 2007a) in which technical interchanges on the architecture are held and project status, schedules, and planning issues are discussed. Alternatively, organizations may utilize cross-site delegation (Bass et al., 2007a) by sending practitioners from remote development sites to the main development site and vice versa.

Prerequisites – Interactions should be planned for and performed regularly. Preference is given to on-site visits. If this appears not to be cost-effective, collaborative technologies such as video-conferencing (Damian and Zowghi, 2002; Clerc et al., 2007b) or wikis (Farenhorst et al., 2007b) could be employed. Furthermore, it is advised to facilitate and improve the decision-making meetings by using a trained human facilitator (Damian and Zowghi, 2002).

Benefits – Frequent interaction helps in maintaining a shared sense of urgency across development sites (Clerc et al., 2007b), which can speed up development time significantly.

Drawbacks – No significant drawbacks are reported in the literature. Although setting up and maintaining the infrastructure imposes some costs, this is of no major influence once the infrastructure is used widely across development sites.
6.3. Architectural Knowledge Management Practices for GSD

**Strategy** – This practice supports a personalization strategy, since frequent interaction helps in identifying ‘who knows what’ among practitioners and interaction does not focus on extracting and capturing architectural knowledge in a repository, but on sharing architectural knowledge among practitioners instead.

### 6.3.2 Cross-site delegation

**Practice Name** – Cross-site delegation (Bass et al., 2007a).

**Intent** – This practice intends to obtain better integration between teams from different development sites by delegation of team members from a local site to a remote site and vice versa (Bass et al., 2007a); delegation involves the physical presence of a team member of one development site to another development site to work and collaborate with that site.

**Motivation** – When establishing a remote site, more effort is needed to align objectives and regard each other as peers. Applying cross-site delegation helps in achieving a shared understanding of the architectural problem that needs to be solved (Clerc et al., 2007b). After the delegation period, the delegate may become a liaison (Clerc et al., 2007b; Bass et al., 2007a).

**Prerequisites** – When applying this practice, it is important that each development site is able to appoint this role to suitable candidates. Consequently, it requires that similar or equal roles (e.g., architects or designers) are present at all development sites.

**Benefits** – Applying this practice results in an increased speed by which architectural knowledge is disseminated to all development sites. Furthermore, applying this practice establishes a means for more effective future interaction and management of architectural knowledge across sites, since practitioners know whom to contact; as such, the problems to initiate contact are alleviated (Hofstede, 2001).

**Drawbacks** – Structural traveling, which is implied by this practice, results in costs related to traveling, housing, and facilities costs. Organizations can use a specialized department to gain efficiency benefits. Furthermore, organizations need to carefully select the roles that travel and the frequency by which traveling occurs (possibly relative to the development phase).

**Strategy** – This practice supports a personalization strategy.

### 6.3.3 Face-to-face project kick-off meeting

**Practice Name** – Face-to-face project kick-off meetings (Damian and Zowghi, 2002).

**Intent** – This practice intends to establish initial relationships across sites.

**Motivation** – It is important to align expectations of practitioners from all development sites involved as early as possible in a software development project to prevent delay because of uncertainties later on.
6. Identifying Practices for AKM in GSD

Prerequisites – A group kick-off meeting with practitioners from various sites could be difficult to plan. Although relevant literature reports on the importance of having face-to-face contact (Damian and Zowghi, 2002; Herbsleb et al., 2005), some other noteworthy results are provided as well, e.g., in a large empirical study (Illes-Seifert et al., 2007): in this study, only 4 out of 55 studied best practices concerns face-to-face communication.

Benefits – A shared understanding of the project’s context and goals can be achieved by sharing architectural knowledge during a kick-off meeting. This helps in speeding up the initial phase of the project and finding the right people across all sites involved in the software development project.

Drawbacks – For large software development projects, a lot of development sites may be involved in determining the architecture for the system to be built. This could lead to planning problems in getting all relevant stakeholders together.

Strategy – This practice supports a personalization strategy.

6.3.4 Urgent request

Practice Name – Urgent request (Bass et al., 2007a).

Intent – This practice intends to quickly collect information on a given architectural topic of interest.

Motivation – During the architecting phase, it can be necessary to, at a given moment in time, collect information on a specific topic in order to proceed with developing the architecture. An example could be the availability of a certain infrastructure component which could be reused across projects. Obtaining a quick response helps in keeping the architecting activities up to speed.

Prerequisites – This practice requires that a distribution mechanism is implemented. The mechanism should be implemented to be separate from general knowledge sharing mechanisms, since a large number of unrelevant requests, or requests for different expertise areas, lowers the sense of urgency and the motivation of the volunteers. In addition, this network of volunteers with expertise on a variety of technical subjects should be created and maintained. Management support and urge for selecting volunteers is important to achieve this. This practice furthermore requires willingness to share information, and thus a sense of openness throughout the organization.

Benefits – Quick responses to urgent and/or ad hoc questions greatly improves the speed by which architectural knowledge is shared across development sites and, consequently, an architecture for a system can be developed. Currently, low-threshold mechanisms to implement this practice exist, e.g., rss feeds (Farenhorst et al., 2007b).

Drawbacks – It is necessary to establish a basic level of trust between the key individuals from different development sites to allow this practice to be successful – one needs to know ‘who is who’ before one will respond to an urgent request.
6.3. Architectural Knowledge Management Practices for GSD

**Strategy** – This practice supports a personalization strategy since it reveals potential sources of architectural knowledge, which then can be accessed through email interaction.

### 6.3.5 Collocated high-level architecture phase

**Practice Name** – Collocated high-level architecture phase \( \text{(Bass et al., 2007a).} \)

**Intent** – This practice intends to create a sound high-level architecture in an efficient way.

**Motivation** – The architecting phase consists of knowledge-intensive tasks and is important for outlining the solution structure for the future development phase. In addition, identifying a high-level architecture generally is done in limited amount of time – certainly compared to detailed design and development activities.

This practice may be implemented by arranging for a single location on which architects from the various development sites site meet. In addition, knowledge management tooling should be catered for to capture the design rationale and major architectural decisions during the phase.

**Prerequisites** – The set of most important quality requirements should be fairly stable. Requirements engineers and domain experts should be available and participate in this phase.

**Benefits** – When organizations apply a collocated high-level architecture phase, all relevant issues can be expected to be tackled as early as possible.

**Drawbacks** – As already described in \( \text{§6.3.3} \), bringing together practitioners from different development sites may lead to planning problems.

**Strategy** – This practice does not support a specific architectural knowledge management strategy.

The practice *collocated initial architecture phase* has been adapted from the requirements engineering domain, where it was termed *collocated analysis phase* \( \text{(Bass et al., 2007a).} \). The practice describes how functional specifications are produced as a result of this collocated analysis phase with a high-level architecture as input. Based on the similarities between requirements and architecture \( \text{(van Vliet, 2008; Hall et al., 2002).} \), we conjecture that a similar practice is applicable for architectural knowledge management in a GSD setting. We are aware that this practice removes the global aspect in GSD by introducing collocatedness. However, this practice does pose additional issues, such as effective sharing of the architectural knowledge of this single location to all development sites, which may be problematic \( \text{(Clerc et al., 2007a).} \).

### 6.3.6 Have a clear organization structure with communication responsibilities

**Practice Name** – A clear organization structure with communication responsibilities \( \text{(Damian, 2007).} \).
6. Identifying Practices for AKM in GSD

**Intent** – This practice intends to maintain open communication lines between well-defined stakeholder roles (Desouza et al., 2006) within the organization. Organizations may be seen as any logical entity of an organization, including departments, teams, and projects.

**Motivation** – Delay can occur when practitioners spend time on finding and reaching the correct person in a given role. Often, it is not the case that information is not available; rather, the information is not shared with the right persons. In earlier research described in Chapter 4 we observed this as well: an organization which applied top-down definition and communication of architecture principles and an organization with more focus on collaborative architecting practices both followed architectural rules; in the former organization, however, architectural decisions did not sink in properly.

This practice may be implemented by creating roles with clear responsibilities and by assigning these roles to the distributed stakeholders. Furthermore, it should be indicated which roles need to communicate with each other in structured meetings.

Other implementation-specific suggestions as proposed by (Illes-Seifert et al., 2007) are: communicating more often, communicating immediately as a question arises (see §6.3.4), communicating according to formal rules (see Chapter 4, using a tool such as a wiki, or using common terminology.

The notion of “organization” may be interpreted differently according to the software architecture and development activities at hand. An organization could include a project organization or a department organization serving multiple projects.

**Prerequisites** – It is important to involve the right people in the meetings that are conducted (Damian and Zowghi, 2003).

**Benefits** – Cross-functional teams associated with developing a specific part of the architecture represent social networks whose members benefit from clear identification of roles and responsibilities in the architectural knowledge management discipline (Damian et al., 2007). It is important to identify these possible social networks in addition to managing the communication responsibilities.

**Drawbacks** – No significant drawbacks are reported in the literature.

**Strategy** – This practice supports a personalization strategy since it helps in defining the structures by which individuals can communicate with each other. This practice does not focus on codification of the knowledge entities that are discussed and exchanged.

### 6.3.7 Have a repository for architecture artifacts

**Practice Name** – Have a repository for architecture artifacts (Damian and Zowghi, 2002).

**Intent** – This practice intends to build a repository to store architectural decisions including the rationale of these decisions.

**Motivation** – Having a shared repository with architectural knowledge helps in obtaining common understanding of the architecture and architectural decisions across development sites and in obtaining awareness of the local working context, as indicated by (Damian and Zowghi, 2002).
6.4 Validation

**Prerequisites** – In order to alleviate the distance in accessing relevant information from a shared repository, it is imperative that all development sites utilize a shared infrastructure. Furthermore, it is important to structure the information in the repository, following e.g., templates as defined by (Tyree and Akerman, 2005).

**Benefits** – When a shared repository is in place, all development sites can easily query and access architectural knowledge.

**Drawbacks** – This practice poses a challenge for the codification strategy: design decisions should be captured (codified) in a way which makes them searchable and reusable in similar situations. Consequently, a starting point for the codification strategy would be to identify appropriate use cases of this architectural knowledge (Clerc et al., 2007c).

**Strategy** – This practice supports a codification strategy.

### 6.4 Validation

We have selected a number of global software development projects from one domain at a large IT service provider (Organization D) to identify to what extent the practices for architectural knowledge management in a GSD setting are applied in practice. We included five projects from different sizes (ranging from 10 until 75 developers) and a different number of development sites (2 until 4 sites, within two countries at most). We performed an initial validation by studying the characteristics of the projects. Following that, we conducted semi-structured interviews with project managers and used the list of practices to collect feedback on the perceived usefulness of the practices. More extensive validation is described in Chapter 7. We provide initial results below:

- Large introduction programmes for new-hires from remote development sites help in obtaining understanding in the context of the organization and specific techniques applied. An off shore desk supports the new-hires in a variety of logistical details to allow for smooth accommodation to the organization’s standards.

- The organization uses common infrastructural platform which makes use of different project-specific and generic environments in which members of projects can interact using collaboration-intensive tools.

- At several levels within the organization, email lists exist on which frequent discussions and questions on a specific topic related to architecture are discussed. These topics are not further structured, but allow for fellow practitioners to share experiences and respond to questions.

- Several projects report that they have traveling plans to allow employees from a remote site to come to the local site.
6. Identifying Practices for AKM in GSD

6.5 Conclusions

Our study aims to identify practices that can overcome global software development challenges in the realm of architectural knowledge management by studying the requirements engineering discipline. Our study so far shows that the practices that have been reported in the requirements management literature can be easily translated to be applicable in the field of architectural knowledge management – we do not see any reason why certain practices would not be applicable. Consequently, we feel that both disciplines can share experiences to learn in understanding each other’s challenges and practices. Furthermore, we conclude that the majority of practices focus on a personalization strategy for architectural knowledge management since the practices support fostering interaction among knowledge workers (Babar et al., 2007). The literature reports on several tools that can help in capturing architectural knowledge and support a codification strategy (Capilla et al., 2007; Babar and Gorton, 2007; Farenhorst et al., 2007a). Based on the results presented in this chapter, we support the arguments from (Desouza et al., 2006; Hansen et al., 1999) that a hybrid approach could combine the best of both worlds.

Initial validation of the architectural knowledge management practices for GSD shows that most of the practices seem applicable in an industrial setting. Our future work is aimed at a further structured validation of the list of architectural knowledge management practices for GSD by collecting detailed feedback on the perceived usefulness of the practices and the actual application of these practices (see Chapters 7 and 9). Furthermore, we plan to augment the list, which is currently primarily based on the literature, with additional practices by focusing on the different types of distance as e.g., identified in (Holmström et al., 2006) to build a balanced set of practices for architectural knowledge management in GSD. The results of this future work is described in subsequent chapters of this thesis.
The Usefulness of Architectural Knowledge Management Practices in Global Software Development

In further strengthening the set of architectural knowledge management practices for global software development, we have identified the perceived usefulness of the practices at a large organization involved in global software development. In this chapter, we report on this study. By including all the Dutch architects from the organization, we obtain a good overview of the perceived usefulness of the architectural knowledge management practices and identify and analyze a peak in the perceived usefulness of the practices at projects that involve three development sites.

7.1 Introduction

Within the field of software engineering, increasing attention is paid to global software development. In GSD, software development takes place at geographically distributed development sites. Although GSD can result in benefits such as reduced development time and increased availability of skilled resources (Carmel, 1999), GSD poses additional challenges as well. Overviews of these challenges have been widely reported (Battin et al., 2001; Ågerfalk et al., 2005; Holmström et al., 2006). We also analyzed these challenges in our earlier research, reported in Chapters 3 and 4; the challenges include lack of informal contact, language differences, and coordination complexity (Ågerfalk et al., 2005).

A specific discipline within the field of software engineering is software architecture. Within the software architecture community, an increasing interest in architectural knowledge management is recognized (Jansen and Bosch, 2005; Kruchten et al., 2006; van der Ven et al., 2006; Babar et al., 2007). Architectural knowledge management involves capturing and communicating the design decisions that lead to a software system, including underlying rationale and context (Avgeriou et al., 2007).

Using architectural knowledge effectively may help in overcoming the challenges and issues encountered in GSD. However, we are currently lacking detailed insight into
architectural knowledge management practices that can effectively be applied in a GSD setting. More specifically, we are interested in identifying the relation between the perceived usefulness of architectural knowledge management practices and the number of sites in a software development project; this will satisfy specific needs of Organization D as well as offer further guidance on applying the architectural knowledge management practices.

To gain insight into the relation between perceived usefulness of architectural knowledge management practices and the number of sites, we build upon previous work in which we identified relevant architectural knowledge management practices based on literature and experience reports from the requirements engineering discipline (see Chapter 6). In this research, we validate these architectural knowledge management practices by conducting empirical research at a large IT service provider in the Netherlands, Organization D.

Our research shows that when the number of sites involved in a software development project increases the perceived usefulness of architectural knowledge management practices does not necessarily increase. The perceived usefulness of architectural knowledge management practices has its maximum in software development projects with three sites. Further investigation revealed that not all projects with three sites are initially set up as such; usually, they start with two sites and evolve to their current situation with three sites. The high perceived usefulness for some architectural knowledge management practices at three sites denotes a need to have these practices implemented proactively.

Furthermore, the usefulness of architectural knowledge management practices in general is confirmed. Yet, some well-known global practices, such as having a clear project structure with communication responsibilities assigned or having a single repository for architecture artifacts, are not perceived as being as useful as one may expect. Finally, our research shows that practices that support a personalization strategy towards knowledge management are perceived as more useful than practices that support a codification strategy towards knowledge management.

This chapter is structured as follows. In §7.2 we provide an overview of related work in the field of software architecture, architectural knowledge management, and global software development. Next, §7.3 lists the research question posed for this research. In §7.4 we describe the approach used for the research. The analysis and the results are described in §7.5 and discussed and validated in §7.6. Finally, §7.7 lists our conclusions.

### 7.2 Related Work

The requirements engineering discipline is one of the first software engineering disciplines in which research has been conducted to identify practices that alleviate the challenges that are involved with global software development. The work of Damian lists requirements engineering challenges and their (proposed and experienced) solutions (Damian, 2006; 2007; Damian and Zowghi, 2003). Bhat et al (2006) provide insight into the challenges with requirements engineering in off-shore outsourced situations and lists ways to overcome these challenges.
7.3. Research Question

A resemblance between a set of requirements for a software system and a set of design decisions taken for that software system has been noted (van Vliet, 2008). In addition, Avgeriou et al. (2007) report on the identification of architectural decisions that pertain to the problem space instead of the solution space. Since the problem space is populated with requirements, the challenges experienced in the requirements engineering discipline in GSD may have their counterpart in the discipline of architectural knowledge management in GSD, as shown by (de Boer and van Vliet, 2009) and Chapter 6. In our research, we combine this topic with other challenges in knowledge sharing in a GSD situation as observed by (Balaji and Ahuja, 2005; Desouza et al., 2006).

In previous work, we gave an overview of the architectural knowledge management practices that can be applied in a GSD setting (Clerc, 2008). We defined a light-weight pattern language for describing the practices which helps us in performing subsequent analyses. Our current research is aimed at validating these architectural knowledge management practices in practice. Farenhorst et al. (2007b,c) list important prerequisites for architectural knowledge sharing and how these prerequisites can be supported by effective tool support. Especially within the collaboration-intensive GSD, these prerequisites are gain in importance. Babar et al. (2007) further focus on different strategies for sharing and managing architectural knowledge, following knowledge management literature: personalization and codification. A personalization strategy promotes interaction between knowledge workers; knowledge is kept by its creator. A codification strategy supports codifying knowledge and storing it in a repository, structured or unstructured. Babar et al. conclude, following (Desouza and Evaristo, 2004), that a hybrid approach between personalization and codification is needed most.

In a workshop on sharing and reusing architectural knowledge (Avgeriou et al., 2007), insight was obtained into the dos and don’ts with respect to architectural knowledge. The workshop participants included both academia and industry. The workshop results show that the majority of the dos pertain to personalization in contrast to codification, but that both knowledge management strategies need to be supported by the service provider’s culture. In addition, the workshop reported other important dos, such as assigning a traveling architect, establishing peers, and allowing for periodic after-the-fact reflection.

7.3 Research Question

For gaining further insight into the architectural knowledge management practices for GSD, we are interested in determining the relation between the number of development sites involved in a global software development project and the perceived usefulness of architectural knowledge management practices. This will further help us to build a set of architectural knowledge management practices with additional guidance on their application. In addition it addresses specific needs of the case study organization as put forward to us. To this end, we formulate our research question as follows:
How does the number of sites involved in a software development project influence the perceived usefulness of architectural knowledge management practices for that project?

7.4 Research Approach

7.4.1 Organization context

We conducted our research at the Dutch branch of a large, international IT service provider, Organization D. The Dutch branch employs over 6,000 employees spread across a number of business sectors, e.g., public, finance, and energy.

Organization D aims for standardization of its software development processes and associated tools. Associated tools include a central configuration management system, standardized electronic project workplaces, and standardized collaboration and communication tooling.

The software development methodology in use by the service provider can be characterized as ‘mixed delivery’. Accordingly, the service provider chooses for an appropriate proportion of activities performed on-site at the customer and works towards an ideal mix between on-site, off-site (at one of the service provider’s offices), near- and off-shored software development.

7.4.2 Approach

We conducted seven semi-structured interviews at prominent multi-site projects at Organization D to collect an initial view of the software development practices that are in place at the organization. The interviews focused on the context of the software development project, the size of the software development project (both in number of project members and workload in terms of necessary man days), and the development methodology. This helped us to construct the questionnaire.

Based on the insight obtained during the semi-structured interviews, we decided to make several changes to the way the architectural knowledge management practices were initially described in Chapter 6. Main reasons for this were to specifically adapt the practices to the jargon of Organization D to create an understandable questionnaire. Hence, for some practices, we decided to reformulate the practice; in these cases, the exact phrase used in the questionnaire is listed between brackets in Table 7.1. We made the following overall changes:

- **Travel to other project locations** – We explicitly added this practice on traveling. Where the practice “cross-site delegation” focuses on traveling to another development to actually do (a substantial amount of) work for a longer period, this practice focuses on traveling for e.g., attending meetings of for project management coordination. Organization D was primarily interested in identifying the best means of project management coordination (e.g., via video-conferencing or via physical visits).

---

1 The organization aims for CMMI level three (Poort et al., 2007).
8 Know who is who across the project (directory) – Organization D was experimenting with developing project intranet sites to contain all information relevant to the project (including all the project stakeholders). Obtaining a view on the perceived usefulness of knowing ‘who is who’ would help in better defining what information should be on these intranet sites. So, after consultation with Organization D, we decided to add this practice.

9 Have a shared infrastructure for work products (e.g., documents, scripts, plans) and source code (configuration management) – Organization D runs several projects with international partners (consortia) in which access to a shared configuration management system was not always possible due to legal constraints. The organization was interested to find out to what extent this was perceived as a problem, and to identify possible differences with non-partnered projects. Hence, we added this practice.

– Collocated high-level architecture phase – The practice to develop the high-level architecture in a collocated manner, a practice distilled from the requirements engineering literature (see Chapter 6), is not directly supported by Organization D or their quality management system. In order not to interfere too much with the management practices implemented at Organization D, we decided to remove this practice.

As a basis for the population of the questionnaire, we decided to use the members of a mailing list to which all Dutch architects of Organization D are subscribed. In this way, we are able to gather information on the complete portfolio of software development projects and thus have variations in the number of sites involved in these projects. In addition, we added relevant roles (architects and designers) from the specific multi-site projects where we conducted the semi-structured interviews. We sent out an email invitation to the architecture mailing list, in which we invited the members to complete the questionnaire.

We asked the participants to complete the questionnaire with a certain (past or ongoing) software development project in mind. Consequently, by indicating the perceived usefulness of a specific architectural knowledge management practice, a respondent does this for the software development project at hand.

Table 7.1 lists the parts of our questionnaire relevant to this chapter. We inquired a list of all the locations where software development occurs for possible future geographic analysis. Furthermore, we decided to use a three-point Likert-scale as described by Jacoby and Matell [1971] and Likert [1932] with values “Yes”, “Neutral or Don’t Know”, and “No”. Specifically, we grouped “Neutral” and “Don’t Know” because we are primarily interested in the significant outlying results, forcing the respondents to a clear statement.
7. The Usefulness of AKM Practices in GSD

Table 7.1: Survey excerpt with those questions relevant to this research

8. Please list all locations where project members were performing software development activities.

10. What practices for architectural knowledge management can be useful within your project? (“Yes”, “Neutral or Don’t know”, “No”)

1. Frequent interaction across sites
2. Cross-site delegation (representatives of each team visit other teams)
3. Face-to-face project kick-off meeting
4. Urgent request (email, mailing list or telephone to quickly get information)
5. Have a clear organization/project structure with communication responsibilities
6. Have a single repository for architecture artifacts (establish a repository for architecture artifacts)
7. Travel to other project locations
8. Know who is who across the project (directory)
9. Have a shared infrastructure for work products and source code (configuration management)

7.5 Analysis and Results

7.5.1 Demographic information

The total population consisted of 363 employees of Organization D. These employees, all members of the architecture mailing list, are performing architecting or architecting-related activities. We received 132 responses, which corresponds to a response rate of 36.4%. Not all of these 132 responses were complete. We selected only the responses that provided answers to the questions that are relevant to our research (see Table 7.1). This resulted in a set of 114 respondents, corresponding to an adjusted response rate of 31.4%. We used these responses as a basis for our research. Although the data does not permit to determine the total number of projects due to confidentiality purposes, analysis of the responses reveals that the responses include at least 70 different software development projects.

The average experience in IT industry of the respondents was 15.2 years. The soft-
ware development projects on average included 48 members. The average duration of the software development project was 193 working days, the average duration of the involvement of the respondents in the software development was 68 working days.

7.5.2 Data preparation

Table 7.2 shows the distribution of the 114 respondents over the number of sites in the software project they were involved in. A site is a separate, geographic location (possibly the customer's location) where a team performs software development activities (Sangwan et al., 2006).

<table>
<thead>
<tr>
<th>No. of sites</th>
<th>No. of respondents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

We regard the number of respondents in each individual category of 4, 5, 6, and 7 sites too low to draw conclusions from. Consequently, we sum up the individual responses of these categories, and form a new group (“4–7 sites”), with 17 responses.

For the analysis of the usefulness of a specific architectural knowledge management practice, we only took into account respondents who rated the perceived usefulness of that practice by answering the appropriate subquestion of question 10. Hence, the response rate per architectural knowledge management practice may differ slightly because not all participants answered all subquestions of question 10 in Table 7.1.

7.5.3 Perceived usefulness of architectural knowledge management practices

Following our research question, we relate the perceived usefulness of each architectural knowledge management practice to the number of sites involved in the software development project of that respondent. We describe our results in the remainder of this section.

We list each practice and start by summarizing the intention of the practice from Chapter 6. Next, we summarize the results of the perceived usefulness of the practice by the respondents. The Appendix shows the responses for each architectural knowledge management practices graphically.

1. Frequent interaction across sites – This practice intends to let practitioners from different sites interact frequently with each other. Interaction may be done through a
7. The Usefulness of AKM Practices in GSD

variety of means, e.g., by using collaboration software such as video-conferencing or wikis, or through on-site, face-to-face interaction. This practice is primarily important for architects to share their views on the architecture with other project stakeholders.

Other research further distinguishes between e.g., audio support and video support (Thomas et al., 2007). Based on the standardization in Organization D’s software development methodology and the combined use of audio support and video support, we have not made this distinction.

We observe a positive attitude towards the usefulness of this practice in GSD. Regardless the number of sites, more than 50% of the respondents regards frequent interaction between the teams involved to be valuable.

2. Cross-site delegation (representatives of each team visit other teams) – This practice intends to obtain better integration between teams from different development sites involved in the software development project by having teams visit each other (e.g., during joint architecture meetings or pair software design or development).

The respondents value this practice, although not as much as some other practices. Furthermore, we observe a clear (positive) peak in the perceived usefulness of this practice at projects that involve three sites, and a lowered value from respondents involved in four or more sites.

3. Face-to-face project kickoff meeting – This practice intends to establish initial relationships across sites by bringing together all project members to hold a joint kickoff meeting.

The respondents clearly value this practice: regardless of the number of sites involved, more than 70% of the respondents value this practice; its perceived usefulness shows a slight increase with the number of sites increasing, but a peculiar drop at three sites.

4. Urgent request (email, mailing list or telephone to quickly get information) – This practice intends to quickly collect information on a given topic of interest.

Organization D has implemented a group mailing address that includes all Dutch architects. The results show that our respondents clearly value this practice; again, more than 70% of the respondents perceive this practice as being useful, regardless of the number of sites. The pattern, however, is different. We observe a drop in perceived usefulness at two sites, and a steady increase towards three and four or more sites.

5. Have a clear organization/project structure with communication responsibilities – As indicated in Chapter we define ‘organization as ‘project’ in the validation of the usefulness of this practice at Organization D, since we are primarily interested in identifying differences across projects in terms of the number of development sites involved in the projects.

We observe that the respondents value this practice less than the previously mentioned practices. We would have expected to observe an increase in perceived usefulness since projects with a higher number of sites often include more project members and a larger variety of roles involved. Instead, we observe that only 30.8% of the respondents who are involved in a project with four sites or more value this practice.

In fact, we sent out our questionnaire using this list.
6. **Travel to other project locations** – This practice intends to obtain better integration between teams involved in the software development project by having architects or architecting roles from different development sites visit each other physically.

   About 40% of the respondents value this practice. One exception exists: 66.7% of the respondents involved in a project with three sites value the practice.

7. **Have a single repository for architecture artifacts (establish a repository for architecture artifacts)** – This practice intends to build a repository to store architectural decisions including the rationale of these decisions.

   The respondents value the perceived usefulness of this practice lowest of all architectural knowledge management practice; at most 40% of the respondents at each site perceive this practice as useful. In addition, we observe a drop in perceived usefulness at the group of respondents working in software development projects that include 4 or more sites. This is reflected by a moderately low correlation of -0.114.

8. **Know who is who across the project (directory)** – This practice intends to allow project members to quickly know who is working within the project. This can be done by providing a directory, consisting of e.g., a mugshot and phone or email contact information.

   We observe no specific impact of the number of sites involved onto the perceived usefulness of this architectural knowledge management practice. This is reflected by a correlation coefficient of -0.041. The perceived usefulness of this practice is not as high as we had expected from previous research; in (Clerc et al., 2007a) we showed that the ‘yellow pages’ containing directory information were the most popular of all system-related information.

9. **Have a shared infrastructure for work products and source code (configuration management)** – This practice intends to have a shared environment or infrastructure where project members can share work products like documents, plans, and source code.

   This architectural knowledge management practice is in general perceived as useful, regardless of the number of sites involved. A prime reason for this may be that this practice is highly supported by the IT service provider’s software development methodology, as described in §7.4.1.

---

### 7.6 Discussion of the Results

#### 7.6.1 Analysis of the number of sites

So far we have presented the perceived usefulness of each architectural knowledge management practice related to the number of development sites. Now we can combine these results to explain differences observed. Fig. [7.1] shows these combined results. We make a number of observations based on these results:

- First, the perception of the usefulness of the architectural knowledge management practices does not differ between projects with one development site (i.e.,
7. The Usefulness of AKM Practices in GSD

![Graph](image)

**Legend**
- Frequent interaction across sites
- Cross-site delegation (representatives of each team visit other teams)
- Face-to-face project kick-off meeting
- Urgent request (email, mailing list or telephone to quickly get information)
- Have a clear organization/project structure with communication responsibilities
- Have a single repository for architecture artifacts (establish a repository for architecture artifacts)
- Travel to other project locations
- Know who is who across the project (directory)
- Have a shared infrastructure for work products and source code (configuration management)

**Figure 7.1:** Overview of the usefulness of AKM practices

local) and projects that involve two sites (i.e., distributed). Only the architectural knowledge management practices ‘using email, mailing list or telephone to quickly get information’ and ‘have a clear organization/project structure’ show a drop of more than 10% in perceived usefulness at two sites. The difference in perceived usefulness of the remaining seven practices does not differ significantly between one or two sites involved.

- Second, we observe a peak in perceived usefulness at software development projects that involve three sites. Only the architectural knowledge management practice ‘frequent interaction’ shows a drop; all other practices show an increase in perceived usefulness. Five out of nine architectural knowledge management practices show an increase of more than 10%.

We have statistically analyzed the observed differences as listed above to verify whether the difference in perceived usefulness observed at software development projects with three sites is significant. We assume that the population follows a normal distribution and applied the chi-squared test (Greenwood and Nikulin, 1996) to test whether the observed variance in the results is statistically significant. To this end, we formulated the following hypotheses:

- $H_0$ – The responses of the group with three sites does not differ significantly from each of the other groups.
- $H_1$ – The responses of the group with three sites differ significantly from each of the other groups.
7.6. Discussion of the Results

For each architectural knowledge management practice, we compared the responses of each group (one site, two sites, three sites, and four or more sites) with the other groups. In general, we observe very high p-values (above 0.5) which result in not being able to discard the null hypotheses; the observed difference in perceived usefulness of a practice per number of sites involved does not appear to be significant. However, we do observe significantly lower p-values (about 0.1) when we observe the usefulness of architectural knowledge management practices focusing on visiting other teams and traveling to other project locations of three sites. Although the p-value observed is still above the 0.05 threshold that is common in empirical research, these lower p-values do urge us to investigate this more thoroughly (see §7.6.3). The main reason for not having obtained statistically significant result is that the data is highly skewed (the number of responses across the different number of sites varies, cf. Table 7.2).

7.6.2 Analysis of architectural knowledge management practices

As with the difference in perceived usefulness of practices depending on the number of sites involved, we combined the results to make some observations. First of all, a difference exists between the perceived usefulness of codification practices and personalization practices. In Chapter 6, we described the codification practice ‘have a single repository for architecture artifacts’ and this research added ‘have a shared infrastructure for work products and source code (configuration management)’, as described in §7.4.2. These codification practices are not perceived as being as useful as the practices that focus on personalization. Rather than a focus on codification, the respondents tend to rely on other forms of communication in which practitioners are linked directly to each other; examples include sharing knowledge during meetings or through other communication mechanisms such as email or mailing lists.

Second, we can relate the observed increase in perceived usefulness of certain architectural knowledge management practices with the patterns observed at other practices. It is important to meet each other at the start of the software development project, during its kick-off. This, in turn, may reduce the importance of knowing ‘who is who’ across the project; a kick-off meeting already caters for this. In addition, when people already know each other, they are more willing to respond to each other’s email or provide each other with information through a mailing list. Hence, we conjecture that a temporal relationship exists between the architectural knowledge management practices (“first do a, because it next reduces the need for b”).

7.6.3 What’s special about three sites?

Analysis of the questionnaire results shows that no significant differences in the perceived usefulness can be identified depending on the number of sites. However, as noted before, we did observe a difference in perceived usefulness of practices at respondents participating in software development projects that involve three sites.

3For this reason, we have chosen not to list all p-values in this section.
In order to analyze the difference, we validated the results described in §7.6.1 and §7.6.2 with key representatives of the IT service provider during Organization D’s annual event for Dutch software and systems engineers.

Of the 102 participants to this event, 15 joined a session that was organized for the purpose of the validation. We presented the findings as listed in §7.6.1 and §7.6.2 and conducted a facilitated group discussion on these findings. We list the results below:

- Projects with four or more sites appear to be more organized. E.g., according to the participants, these projects need to organize communication across sites, because it will be more complex and expensive when it is not organized. Hence, communication often occurs through a ‘hub-and-spoke’ model, which results in complexity \( O(n) \) instead of \( O(n^2) \), where \( n \) is the number of sites involved.

- Traveling does not scale when the number of sites involved in the software development project increases.

- Software development projects are initially thought to be set up either with more than three sites or with only one or two sites. Projects with three sites typically evolved to that stage in the course of the project, i.e., they started as a one- or two-site project. Following that, the main conclusion of the participants of the validation session is that software development projects with three sites organize work in a different way than projects with a different number of sites.

To validate the above statements, we directly contacted the 18 respondents that were involved in a software development project with three sites and the 11 respondents involved in projects with four sites. We asked them a number of questions to investigate further characteristics of these projects:

- How was the software development project organized when it started? What was the size and number of sites at the start?

- How is work currently divided in the software development project? Is there a central site that divides the work across the three sites?

We collected responses from three different software development projects that involved three sites and one project that involved four sites.

We learned that two software projects with three sites did not start as such but evolved to that stage after having run for several months with two sites. The two sites that were involved at the start of the project included the customer’s location and one of the development centers of Organization D near the customer’s location. After a couple of months, project management decided to move software development activities to the service provider’s offshore development centers. At that time, communication was not structurally reorganized, which resulted in less communication between the sites. In addition, the practitioners involved in software projects with three sites indicated an increased perceived usefulness for some architectural knowledge management practices because they encountered several problems in their project. The practitioners indicated that the architectural knowledge management practices were not implemented
but, rather, thought that applying these practices would help in addressing these problems.

Conversely, the software development project with four sites was designed as such at the start of the project and remained stable in number of sites since then. Most architectural knowledge management practices were either implemented or discarded on beforehand; e.g., traveling does not scale with four geographically distributed development sites, so other measures had already been taken.

In conclusion, software development projects that encounter an increase in number of sites should be keen to implement additional architectural knowledge management practices in time to prevent problems from surfacing.

### 7.6.4 Limitations

We list possible limitations to our study by analyzing the internal validity and external validity following (Kitchenham et al., 2002; Wohlin et al., 2002; Runeson and Höst, 2009).

**Internal validity** – Numerous other factors than the number of sites of a software development project may affect the perceived usefulness of architectural knowledge management practices. Although in this research we primarily focus on the number of sites, we used the validation session at Organization D to identify several other, underlying reasons, as described in §7.6.3.

**External validity** – We conducted our questionnaire at one organization. We asked the respondents selected from this organization to answer the questionnaire with a certain software development project in mind. When this questionnaire is repeated over time, the same group of respondents may be involved in different software development projects and answer the questions according to that project. Given the nature of our population (the complete population of Dutch architects of the service provider) and the variety of software development projects encountered in this questionnaire, we do not expect significant deviations. However, when the questionnaire is completed by another organization, different results may be obtained.

### 7.7 Conclusions

We have conducted empirical research on the relation between the usefulness of architectural knowledge management practices in global software development. Specifically, we related the usefulness of these practices to the number of sites involved in several GSD projects at a large IT service provider in the Netherlands. We sent out a questionnaire to the Dutch architects of this organization to obtain our results. Using the questionnaire, we related the number of sites to the perceived usefulness of nine practices for architectural knowledge management in GSD, most of which were identified in earlier research (described in Chapter 6). Seven of these nine practices support a personalization strategy towards knowledge management, the remaining two support a codification strategy towards knowledge management.
The architectural knowledge management practices in general are perceived as useful (about 40% of the respondents regards the architectural knowledge management practices for GSD as useful). Personalization practices in general are perceived as more useful than codification practices. Some architectural knowledge management practices show an increase in usefulness as the number of sites increases. This increase peaks at software development projects that involve three sites. Other practices are regarded as useful regardless of the number of sites involved. We learned that the main reason for the increase is that practitioners involved in projects with three sites have a need for additional architectural knowledge management practices because certain problems are encountered. Several three-site software development projects were not started with three sites but evolved into that stage after having run for some time with two sites. The high perceived usefulness for some architectural knowledge management practices in three-site software development projects denotes a need to have these practices implemented proactively to prevent problems with architectural knowledge sharing from arising.
In this chapter we report on a study in which we investigated how the practices for architectural knowledge management in global software development can be strengthened by adequate tool support. In this study we have looked into wikis and wiki functionalities and describe how generic wiki functionalities can be utilized to implement (part) of the practices. Furthermore, we show how we have implemented three use cases that can support the architectural knowledge management practices in global software development in a prototype semantic wiki implementation. This study has brought us the insight that wikis, and semantic wikis in particular, are powerful means to capture architectural knowledge and reason with (query, select) the combined set of architectural knowledge elements to address stakeholders concerns or perform architectural design activities.

8.1 Introduction

In the past few years, the software architecture community has shown an increasing interest in architectural knowledge, see e.g., Jansen and Bosch (2005), Kruchten et al. (2006), Babar et al. (2007, 2009), Farenhorst and de Boer (2009). We define architectural knowledge as the integrated representation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context and environment. Sound management of architectural knowledge is important since it facilitates a better decision-making process in shorter time, saving re-work and improving the quality of the architecture (Babar et al., 2007; Rus and Lindvall, 2002).

In global software development, software engineering practices are performed at geographically separate locations. With the trend of GSD, the management of architectural knowledge becomes even more important due to challenges that arise as a result of the geographical, temporal, and socio-cultural distance involved with GSD.
Overviews of the challenges in GSD have been widely reported (Battin et al., 2001; Ågerfalk et al., 2005; Holmström et al., 2006) and include the lack of informal contact, linguistic differences, and coordination complexity. Furthermore, (Al-Ani and Redmiles, 2009) show that the decision-making-process in GSD typically is undocumented and hence runs implicit. Al-Ani and Redmiles show that regardless of team size or distribution, the decision-maker is the leader of the team in which the decision-making-process runs. In other words, sound architectural knowledge management is also affected by organizational and personal culture and tools alone do not provide a panacea.

In our research, we focus on using architectural knowledge effectively to overcome the challenges associated with GSD. We have selected and developed a number of practices for architectural knowledge management that help to overcome these GSD challenges, see Chapter 6. Although we have performed a validation of the usefulness of the architectural knowledge management practices (see Chapters 6 and 7), we envision wider tool-based support to implement the practices effectively.

We have gained confidence that wikis are a useful means to manage architectural knowledge, see e.g., Farenhorst et al. (2007a; Farenhorst and van Vliet, 2008). A wiki is a “collaboratively created and iteratively improved set of web pages, together with the software that manages the web pages” (Wagner, 2004), based on Leuf and Cunningham (2001). Yet, we currently lack insight into the extent to which wikis can be used to support architectural knowledge management in a global setting.

To address this lack, our research focuses on identifying generic functionalities implemented in wikis and on identifying how current practices for architectural knowledge management in GSD can be implemented using these functionalities.

The results show that wikis offer substantial functionality for implementing architectural knowledge management practices in GSD; yet, wikis alone are not enough for implementing architectural knowledge management for GSD. Three of the identified practices, mainly supporting a codification strategy towards knowledge management, can be implemented completely. Another three architectural knowledge management practices for GSD, mainly supporting a personalization strategy towards knowledge management, can be implemented partially, and one practice cannot be implemented using wiki functionalities. Furthermore, because several distinct wiki functionalities can be used to implement the architectural knowledge management practices, we conclude that a hybrid approach including a codification and a personalization strategy towards architectural knowledge management is beneficial in GSD.

This chapter is structured as follows. First, §8.2 lists the research questions for this research and describes the approach we used. Next, §8.3 summarizes the generic wiki functionalities. In §8.4 we describe the architectural knowledge management practices for GSD and how these practices can be implemented using wiki functionalities. In §8.6 we provide our conclusions and we conclude this chapter with a discussion on the implications in §8.7.
8.2 Research Question and Approach

Over the past few years, the workshop series on *Learning Software Organizations* explored the use of wikis for knowledge management in software engineering, see e.g., [Chau and Maurer, 2004]. More recently, specific proprietary wiki implementations have been developed for capturing architectural knowledge, e.g., [Zimmermann et al., 2009]. In our research, we build upon the insights reported and are interested in identifying to what extent our architectural knowledge management practices specifically for GSD can be supported using wikis. Wikis have a broad applicability and to further broaden the applicability, we do not want to limit ourselves to specific wiki implementations. Hence, we first identify a set of generic functionalities that are typically present in wiki applications. Using this list of wiki functionalities, we next determine for each architectural knowledge management practice on which wiki functionality or functionalities it can be built.

8.2.1 Research question

Following this, we formulate our central research question as follows:

*What functionalities of wikis can be employed to implement architectural knowledge management practices for global software development?*

8.2.2 Research approach

To identify possible wiki functionalities, we performed a critical analysis of the literature. We used the IEEE Digital Library and ACM Portal and searched for papers whose abstracts or titles contained both the terms “wiki” and “functionalities”, since this directly follows from our research question. Following [Brereton et al., 2007], we constructed different, resource-dependent searches in the libraries. From the resulting set of twenty papers, we read the titles, abstracts, and conclusions and found that a substantial set of papers concerned specific wiki implementations and did not focus on generic wiki functionalities. If the abstract and conclusions of a particular paper discussed wikis in general and listed functionalities, we selected that paper as a basis for our research.

For selecting the architectural knowledge management practices for global software development, we used our existing work as a basis for selecting the practices. In addition, we applied the approach described above and selected other relevant articles from the IEEE and ACM Digital Libraries using the keyword “architectural knowledge management”. This resulted in 36 papers matching our search criteria. Of these papers, we read the abstract and conclusions, as described above.
8. Supporting AKM in GSD

8.3 Wiki Functionalities

Although wikis build upon web technologies that have been around for quite some years, wikis offer the ability to change and refine the content of web pages immediately; as such, wikis augment generic web functionalities and offer new collaboration possibilities.

For our research, we are interested in defining generic wiki functionalities. For the literature research performed to elicit these functionalities, we selected the papers listed below as a basis for generic wiki functionalities. From these literature sources, we deducted the generic wiki functionalities and combined and merged terminology if terms differed across the sources.


1. **Collaborative authoring** (Parker and Chao, 2007) – Wiki users can collaboratively create documents or review and/or edit wiki content created by others. As such, all wiki users are both reader (consumer) and contributor (producer) of the information stored in wikis. Furthermore, wiki content is not static in nature, but rather dynamic: several versions of the wiki content (wiki pages) can exist. Collaborative authoring is possible because wikis are open systems in nature and allow the content to be constructed incrementally.

2. **Authentication of users** (Ebersbach et al., 2008; Parker and Chao, 2007) – Authentication is the process of determining whether someone actually is who it is declared to be. Usually, wikis implement authentication using username/password pairs. Authentication allows wiki administrators to prevent certain users from editing or viewing wiki content (applying authorization).

3. **Profile pages** (Augar et al., 2004) – Functionality that is related to the “Authentication of users” allows users to create profile pages. On these pages, wiki contributors

---

1Using other functionality (i.e., “Authentication of users”), reading and/or modifying wiki content may be restricted (Ebersbach et al., 2008).
can describe themselves and their expertise areas, which may facilitate (architectural) knowledge sharing even further.

4. History (or (back-)tracking) (Ebersbach et al., 2008; Leuf and Cunningham, 2001; Parker and Chao, 2007) – Wikis offer the possibility to roll back changes that were made to wiki content by storing all previous versions of the content. As such, this functionality makes it possible to revert to status of the wiki at any given moment in time.

5 Search functionality (Ebersbach et al., 2008) – Wikis offer full-text or title search to allow for quick access to wiki content.

6. Share information among participants (Leuf and Cunningham, 2001) – Wikis, as other web-based systems, offer the functionality to share (possibly collaboratively authored) content to other users of the wiki (Parker and Chao, 2007). Specific mechanisms that implement this in wikis include rss feeds or the possibility to subscribe to changes on designated wiki content.

7. Create a multi-media base (Parker and Chao, 2007) – Wikis allow for collaboration between groups (e.g., project members) by storing information such as documents, presentations, audio/video content, and meeting minutes as appendices (attachments) on wiki pages. Furthermore, wiki users can use rich text functionality in individual wiki pages to augment text and enhance information density.

8. Create trails between pages (Reinhold, 2006) – Wikis can offer a wide range of interlinked content (pages). With large numbers of pages, and various links between these pages back and forth, it may become difficult for someone to find one’s way through the content. Wiki trails address this issue by offering the possibility to define paths or trails that suit the information needs for specific stakeholders.

### 8.4 Implementing Architectural Knowledge Management Practices for Global Software Development with Wiki Functionalities

In this section we describe a list of architectural knowledge management practices for global software development and elaborate on the wiki functionalities that can be used to implement each of these practices. First, we give a brief overview of the architectural knowledge management practices for GSD that were distilled from our previous research and additional literature. Next, shows how we can map the architectural knowledge management practices for GSD onto generic wiki functionalities.

#### 8.4.1 Architectural knowledge management practices for global software development

This section lists the literature sources we used as a basis to elicit the architectural knowledge management practices for global software development.
8. Supporting AKM in GSD


We selected architectural knowledge management practices for GSD from the literature sources by identifying best practices or experiences that were reported. Initial theories that were not validated at all were not selected as a basis for our work.

A. **Have a repository for architecture artifacts and architectural decisions** (Clerc, 2008) – This practice intends to build a repository to store architectural decisions including the rationale of these decisions. Having a shared repository with architectural knowledge helps in obtaining common understanding of the architecture and architectural decisions across development sites and in obtaining awareness of the local working context. Furthermore, expanding or linking existing knowledge bases from different sites can prove beneficial; existing knowledge bases can be connected, creating a custom shell with preloaded knowledge (de Boer et al., 2007; Neches et al., 1991).

B. **Have a clear organization structure with communication responsibilities across sites** (Clerc, 2008) – This practice intends to maintain open communication lines between well-defined stakeholder roles. This practice may be implemented by creating roles with clear responsibilities and by assigning these roles to the distributed stakeholders. Furthermore, it is important to indicate which roles need to communicate with each other in e.g., structured meetings.

C. **Share relevant architectural knowledge to all sites** (Clerc, 2008; de Boer et al., 2007) – When software development occurs geographically distributed, the need arises to share information between different stakeholders on different sites. Software development professionals require awareness of the architectural knowledge needed to perform their work effectively. Knowing who knows what (e.g., using skills management (Rus and Lindvall, 2002) or ‘yellow pages’ (Clerc et al., 2007a)) can prove to be beneficial.
8.4. Implementing AKM Practices for GSD with Wiki Functionalities

D. Urgent request (or: quickly collect information on an architectural topic of interest) *(Clerc, 2008)* – During the architecting phase, it can be necessary to, at a given moment in time, collect information on a specific architectural topic in order to proceed with developing the architecture (e.g., the availability of a certain infrastructure component which could be reused across projects). Obtaining a quick response helps in not deterring design activities.

E. Propose and rank alternatives when taking decisions *(de Boer et al., 2007)* – Constructing an architectural design essentially focuses on taking well-founded architectural decisions based on a ranking of one or more alternatives considered. This decision-making process in fact is the core of architectural knowledge management.

F. Ensure traceability of architectural knowledge *(de Boer et al., 2007)* – One of the important possible uses of architectural knowledge is the ability to trace back architectural decisions made and identify what architectural concerns lied at the basis of the architectural decision and what alternatives were considered for that decision. Lowered traceability of architectural knowledge has led to probable causes for problems with architectural knowledge management such as an invisible decision-making process to different sites and other knowledge sharing problems.

G. Frequent interaction across sites *(Clerc, 2008)* – In GSD, the geographical distance and possible time difference burden effective sharing of architectural knowledge. This may result in language and terminology problems. More frequent interaction across sites helps to address these problems. In addition, more frequent interaction help in building trust across sites *(Clerc et al., 2009)*.

8.4.2 Mapping architectural knowledge management practices for global software development onto wiki functionalities

Now that we have provided an overview of generic wiki functionalities in §8.3 and a list of architectural knowledge management practices for global software development in §8.4.1, we now show how we can map the practices onto the functionalities. This mapping was constructed using the insights gained in performing the critical analysis of the literature for wiki functionalities and architectural knowledge management practices for GSD.

Table 8.1 shows how wiki functionalities can be used to implement architectural knowledge management practices in GSD. In this figure, an arrow between an architectural knowledge management practice and a wiki functionality indicates that the architectural knowledge management practice can be implemented with the designated wiki functionality. We describe each of relations between architectural knowledge management practices for GSD and associated wiki functionalities in turn, from the perspective of the architectural knowledge management practices.

A ‘repository for architecture artifacts and architectural decisions’ can be implemented on a wiki because (codified) architectural knowledge can be stored in the wiki using the functionality to create a multi-media base. Using this multi-media base, it is possible to store text, graphics, audio, and video fragments, all of which may contain architectural knowledge. Hence, wikis can serve as a repository for architectural
8. Supporting AKM in GSD

Table 8.1: Relating architectural knowledge management practices for global software development and wiki functionalities

| A. Have a repository for architecture artifacts and architectural decisions | x  | x  |
| B. Have a clear organizational structure with communication responsibilities across sites | x  | x  |
| C. Share relevant architectural knowledge to all sites | x  | x  |
| D. Quickly collect information on an architectural topic of interest | x  | x  |
| E. Propose and rank alternatives when taking decisions | x  | |
| F. Ensure traceability of architectural knowledge | x  | x  | x  |
| G. Frequent interaction across sites | x  | x  | x  |

Furthermore, to be able to search the repository effectively, search functionalities of wikis can be employed. We acknowledge that even more structure in a wiki (e.g., using a semantic wiki) can be even more helpful in excavating architectural knowledge from the knowledge base using reasoning rules and automated querying. For a prototype and a further discussion, see §8.5.2 and §8.7.

A ‘clear organizational structure with communication responsibilities across sites’ can be partially implemented by using the wiki functionality to create profile pages for users and/or groups detailing on their responsibilities. Moreover, authentication of users to parts of the wiki content can be used to implement communication structures akin to the organizational structure that was set up. Architects, for example, can distribute the rights to edit architectural knowledge on the wiki to designated groups of people (e.g.,
senior developers) and thus protect the architectural knowledge from manipulation by others. Yet, additional effort is necessary to install and empower the organizational structure and the stakeholders in their responsibilities. A (project) manager should cater for this.

The architectural knowledge management practice to ‘share architectural knowledge to all sites’ can obviously be implemented by sharing information to all participants. When content is provided on the wiki, the pull mechanism is employed where stakeholders visit the wiki to consume the knowledge. Alternatively, *rss feeds* can be employed to push relevant architectural knowledge to certain stakeholders, or to notify subscribed stakeholders if new architectural knowledge emerges. With such a push mechanism implemented effectively, users don’t need to search the wiki themselves (Farenhorst et al., 2007b; Lago et al., 2010).

To ‘quickly collect information on an architectural topic of interest’ the information first of all needs to be made available. This can be effected by allowing stakeholders to collaboratively capture architectural knowledge on the wiki. Adequate search functionality, in addition, may help stakeholders to search for architectural knowledge (see the aforementioned description on the repository for architectural knowledge). This practice, however, can only be fully implemented when stakeholders respond quickly to a request for information that currently is not captured in the wiki content; wikis cannot help in this. Hence, this architectural knowledge management practice for GSD can be only partially implemented using wiki functionality.

Using the inherent wiki functionality of collaborative authoring, it is possible to ‘propose and rank alternatives when taking architectural decisions’. However, wiki functionality does not allow to rank alternatives automatically; this requires expert knowledge of the architect – knowledge that in turn may be captured on the wiki to enrich the base with architectural knowledge.

The practice ‘traceability of architectural knowledge’ involves understanding of the alternatives that were considered when taking an architectural decision. This practice can be implemented using the history or (back-)tracking functionality of wikis. Earlier versions of wiki content in which the architectural decision is captured allow for a *replay* of the decision-making process over time, including identifying how the need for the decision originated and what stakeholders were involved in taking the decision. Moreover, the search functionality of wikis can help in excavating the architectural knowledge. Apart from that, traceability of architectural knowledge can be implemented using a structure such as a wiki template or semantic wiki (see §8.5 and §8.7).

The architectural knowledge management practice for GSD ‘frequent interaction across sites’ is not covered by any of the generic wiki functionalities. Wikis nor any other technological functionality can urge a GSD organization to interact more frequently. Of course, useful technological means (e.g., videoconferencing or chatting) can be employed when the organization has chosen how to implement frequent interaction. We regard it as the responsibility of the project manager to cater for adequate frequent interaction across the sites involved in the GSD project.
8. Supporting AKM in GSD

8.5 Implementing a Semantic Wiki

The research described thusfar in this chapter provides us with confidence that wikis are a good means to support the architectural knowledge management practices for GSD. To further substantiate the results obtained in this study, we have developed a prototype implementation of an architectural knowledge sharing environment using a semantic wiki. Semantic wikis extends wiki flexibility by allowing for reasoning with structured data: semantic annotations to that data correspond to an ontology that defines certain properties (Schaffert et al., 2008; Liang et al., 2009). de Boer and van Vliet (2011) reports on experiences in introducing semantic wikis for architectural knowledge management in e-government and distributed development.

For the prototype, we developed a software engineering ontology and implemented three use cases for architectural knowledge using this ontology. The use cases are described in §8.5.1. Next, §8.5.2 describes the implementation of the use cases in our prototype, SE-Wiki.

8.5.1 Use cases implemented in a semantic wiki

This section elaborates on three use cases for architectural knowledge. The use cases are concrete elaborations of the use cases described in Chapter 2. The use cases can be deployed in global software development organizations in that it builds upon knowledge that is readily available in the semantic wiki, regardless the development site at which it is created. Furthermore, the use cases are centered around the architect, a potential rotating or distributed role.

The use case of software reuse supports the architect to if existing software can be reused to implement a new functional requirement. The use case changing requirement supports the architect in updating an architecture design according to a changing requirement. The use case design impact evaluation supports the architect in evaluating and identifying the impact of changing requirements on architecture design. The use cases are described following a technique introduced in (Lago et al., 2010). The use case scenario is described using a problem and solution description and a detailed description of the scenario.

Scenario 1 – software reuse

An architect wants to check if existing software can be reused to implement a new functional requirement, and the new functionality is similar to the existing functionality.

Problem – The architect needs to understand the viability of reusing software to satisfy existing and new functional and quality requirements.

Solution – The architect first finds all the architecture components that realize the existing functional requirements which are similar to the new functional requirement. Then, the architect can trace the existing architecture components to determine what quality requirements may be affected and if the existing software supports the new requirement.

This section is based upon joint work with Antony Tang and Peng Liang (Tang et al., 2011).
Scenario description

1. The architect thinks that the existing software can support a new functional requirement which is similar to existing functional requirements.

2. The architect selects the existing functional requirements and identifies all software components that are used to realize them.

3. For each software component found, the architect identifies the related architectural structure and the quality requirements.

4. The architect assesses if the existing quality requirements are compatible with the quality requirements associated with the new functional requirement.

5. If so, the architect decides to reuse the components to implement the new functional requirement.

Scenario 2 – changing requirement

An architect wants to update the architecture design because of a changing functional requirement.

Problem – The architect needs to understand the original requirements and the original architecture design in order to cater for the change.

Solution – The architect first finds all existing requirements that are related to the changing requirement. Then the architect identifies the decisions underlying the original design. The architect can assess how the changing requirement would affect related existing requirements and the original design.

Scenario description

1. The architect identifies all the related artifacts (e.g., related requirements, architectural design decisions, and design outcomes) that concern the changing requirement.

2. The architect evaluates the appropriateness of the changing requirement with related existing requirements.

3. The architect extracts previous architectural design decisions and rationale for the changing requirement.

4. The architect identifies new design issues that are related to the changing requirement.

5. The architect proposes one or more alternative options to address these new design issues.

6. The architect evaluates and selects one architectural design decision from the alternative options. Criteria for the evaluation include that the selected decision should not violate existing architectural design decisions and that the selected decision should satisfy the changing requirement.
7. The architect evaluates whether the new architectural design outcome can still satisfy those non-functional requirements that are related to the changing functional requirement.

Scenario 3 – design impact evaluation

An architect wants to evaluate the impact a changing requirement may have on the architecture design across versions of this requirement.

**Problem** – The architect needs to understand and assess how the changing requirement impacts the architecture design.

**Solution** – The architect finds all the components that are used to implement the changing requirement in different versions and evaluates the impact of the changing requirement on the architecture design.

**Scenario description**

1. The architect extracts all the components that realize or satisfy the changing requirement in different versions, functional or non-functional.

2. The architect finds all the interrelated requirements in the same version and the components that implement them.

3. The architect evaluates how the changes between different versions of the requirement impact on the architecture design. Furthermore, the architect can recover the decision made for addressing the changing requirement.

8.5.2 Prototype implementation: SE-Wiki

The use cases described in §8.5.1 have been implemented in a semantic wiki, termed SE-Wiki. In the semantic wiki, we implemented an ontology to specifically support co-evolving architectural requirements and design (see Tang et al., 2011, Fig. 2). Furthermore, we describe how the prototype supports the use case scenarios by using the example of the development of a Portal.

**Scenario 1 – software reuse**

**Description** – An architect wants to check if existing software can be reused to implement a new functional requirement which is similar to existing functional requirements that have been implemented (see §8.5.1).

**Example** – A new functional requirement *Change User Access: The Portal should be able to change user’s access rights to resources* is proposed by the Portal Manager. The architect thinks that this new functional requirement is similar to an existing functional requirement, i.e., *Track Usage: The Portal tool should be able to track usage of resources by all users*. The architect wants to check if the existing software (i.e., design

---

3Resources in the project under study refer to all the information maintained by the Portal.
8.5. Implementing a Semantic Wiki

outcomes/architecture) that is used to implement the requirement *Track Usage* can be reused to implement the new requirement *Change User Access*, especially with regard to the quality requirements.

Since the requirements and architecture specifications are already semantically annotated in *SE-Wiki*, a semantic query can be employed to query the direct and indirect tracing relationships (see Tang et al., 2011, Fig. 2) from an instance of *Functional Requirement* (i.e., the existing functional requirement *Change User Access*) to all the concerned *Design Outcomes* that realize this functional requirement, and all the *Non-Functional Requirements* that the *Design Outcomes* can satisfy. The snapshot of this scenario through a semantic query is shown in Fig. 8.1. The top part of this figure is the editing box for the semantic query input and the lower part shows the query results.

![Semantic search interface](image)

**Figure 8.1:** Scenario 1 through the semantic query interface in *SE-Wiki*

**Scenario 2 – changing requirement**

**Description** – An architect wants to update an architecture design because of a changing functional requirement (see §8.5.1).

**Example** – A functional requirement *Change User Access: The Portal tool should be able to change user’s access rights to resources* is changed into *Change User Access: The Portal tool should only allow a System Administrator to change user’s access rights to resources*. Accordingly, the design based on this requirement should be updated as well. To achieve this, the architect should make sure that this changing requirement has no conflict with related existing requirements, and understand the context of this requirement before updating the design. The architect then extracts all the related artifacts concerning this changing requirement by navigating to the wiki page of this requirement in *SE-Wiki*. The wiki page records all the artifacts (e.g., requirements, architectural design decisions, and design outcomes) related to this requirement as shown in Fig. 8.2.
8. Supporting AKM in GSD

**Change User Access**

*Requirement Description:* The Portal tool should be able to change user’s access rights to resources.

*Requirement ID:* FR-006

*Proposed by:* Network Manager

*Stakeholder:* Functional Requirement

<table>
<thead>
<tr>
<th>Decision</th>
<th>Depend on</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portal Personalization</td>
<td>Change User Access</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Qual is related to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration Requirement</td>
<td>Track Usage, Change User Access</td>
</tr>
<tr>
<td>Interoperability Requirement</td>
<td>Track Usage, Change User Access</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design Outcome</th>
<th>Realizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Web Page</td>
<td>Change User Access</td>
</tr>
<tr>
<td>REST Structure</td>
<td>Change User Access</td>
</tr>
<tr>
<td>SOA Structure</td>
<td>Change User Access</td>
</tr>
</tbody>
</table>

*Category:* Functional Requirement

---

Figure 8.2: Scenario 2 through an in-line semantic query in SE-Wiki

**Scenario 3 – design impact evaluation**

**Description:** Requirements are frequently changed from one software version to the next. An architect tries to evaluate and identify the impact of the changing requirements on architecture design, so that requirements and architecture design can be kept consistent.

**Example:** The requirement *Change User Access* is updated in the next version, i.e.,

*Version 1:* The Portal tool should be able to change user’s access rights to resources, and

*Version 2:* The Portal tool should only allow System Administrator to change user’s access rights to resources. The architect extracts different versions of the requirement with the same requirement ID using a semantic query (e.g., *[Category:Requirement][is identified by::DC 001]*), in which *DC 001* is the DC element to identify the version of a requirement (see Tang et al., 2011, Fig. 2). The architect finds the components for implementing the requirements by navigating to the wiki page of the requirement in different versions. The architect then finds the other components for implementing related requirements through reasoning support (e.g., iteratively traverse all the related requirements), which is based on the reasoning rules and relationships defined in the ontology. Using the information available to him, the architect can identify the changes to the architecture design in two sequential versions of the requirement. From that, the architect can evaluate the change impacts to the architecture design. Fig. 8.3 shows the comparison of the wiki pages of a requirement across two versions (the left hand side shows the latest version of the requirement *Change User Access*, and the right hand side shows a previous version of *Change User Access*, which is superseded by the latest version). The requirement changes between versions with changed decisions and design...
8.6 Conclusions

In this chapter we have investigated how practices for architectural knowledge management in global software development may be implemented using (semantic) wikis. To this end, we have mapped the architectural knowledge management practices for GSD onto a list of generic wiki functionalities distilled from the literature. Furthermore, we have implemented several use cases for architectural knowledge in a prototype semantic wiki implementation, SE-Wiki.

The results show that wikis offer substantial functionality for implementing architectural knowledge management practices in GSD. Three of the identified practices can be implemented completely, another three can be implemented partially, and one practice cannot be implemented using wiki functionalities. The following architectural knowledge management practices for GSD can be implemented completely using wiki functionalities: ‘have a repository for architecture artifacts and architectural decisions’, ‘share relevant architectural knowledge to all sites’, and ‘traceability of architectural knowledge’. These practices mainly support a codification strategy towards knowledge management. Furthermore, the following architectural knowledge management practices for GSD can at most be partially implemented using wiki functionality: ‘have a clear organizational structure with communicating responsibilities across sites’, ‘quickly collect information on an architectural topic of interest’, and ‘propose and

---

**Change User Access**

<table>
<thead>
<tr>
<th>Requirement Description</th>
<th>System Administrator to change user’s access rights to resources.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement ID</td>
<td>FR-006</td>
</tr>
<tr>
<td>Requirement is related to</td>
<td>Track Usage</td>
</tr>
<tr>
<td>Added decision</td>
<td>Portal Personalization</td>
</tr>
<tr>
<td>Depend on</td>
<td>Change User Access</td>
</tr>
<tr>
<td></td>
<td>Change User Access 001</td>
</tr>
</tbody>
</table>

**Category: Functional Requirement**

---

**Change User Access 001**

<table>
<thead>
<tr>
<th>Requirement Description</th>
<th>The Portal tool should only allow System Administrator to change user’s access rights to resources.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement ID</td>
<td>FR-006</td>
</tr>
<tr>
<td>Requirement is related to</td>
<td>Track Usage</td>
</tr>
<tr>
<td>Added decision</td>
<td>Portal Personalization</td>
</tr>
<tr>
<td>Depend on</td>
<td>Change User Access</td>
</tr>
<tr>
<td></td>
<td>Change User Access 001</td>
</tr>
</tbody>
</table>

**Category: Functional Requirement**

---

Figure 8.3: Scenario 3 through comparison in SE-Wiki
8. Supporting AKM in GSD

rank alternatives when taking decisions’. These practices mainly have a personalization strategy towards knowledge management, since it urges the knowledge workers to interaction with each other. Finally, the architectural knowledge management practice for GSD ‘frequent interaction across sites’ cannot be implemented using wikis. Wikis may be used in interacting across sites, but setting up (implementing) this practice requires planning of meetings and travels, which is up to project managers.

We furthermore conclude that several distinct wiki functionalities can be used to implement the architectural knowledge management practices: first of all, functionality to author and share architectural knowledge and functionality to track and trace wiki content. This functionality supports a codification strategy towards knowledge management. Second, profile pages and authentication support a personalization strategy towards knowledge management; this functionality does not cater for capturing architectural knowledge itself, but is used to provide information that supports interaction between knowledge workers. Hence, we conclude that a hybrid approach including a codification and a personalization strategy towards architectural knowledge management is beneficial in GSD (Babar et al., 2007; Hansen et al., 1999).

Our prototype implementation SE-Wiki supports a traceability metamodel and implements several traceability use cases using a traceability ontology. Furthermore, SE-Wiki supports semantic annotation and traceability, and the annotated semantic wiki pages provide an information base for constructing semantic queries. These features support the architect in reasoning with architectural knowledge, independent of location, time zone, or socio-cultural borders. Once the architectural knowledge is captured in the semantic wiki, it may help in achieving the benefits of GSD.

Unfortunately, we have not been able to validate the usefulness of the prototype semantic wiki in a real-life example, e.g., via a study at one of the industrial partners in our research project. Yet, given the informal feedback we did receive and experiences reported by (de Boer and van Vliet, 2011), we believe that the prototype is viable and can be effective. Hence, our prototype provides a valuable addition to address GSD challenges by allowing practitioners to effectively use architectural knowledge.

In our earlier research, we have shown that a sound communication structure which caters for communication across sites is important when distributing work and knowledge across sites (see e.g., Chapters 3 and 4). Implementing this communication structure is part of one of the architectural knowledge management practices for GSD but in fact has implications for all other architectural knowledge management practices considered in this research.

8.7 Implications

The work described in this chapter has a number of implications. First of all, when populating a wiki with architectural knowledge, adhering to a structure may prove beneficial. This structure helps in aligning various stakeholders with different cultural and organizational backgrounds. Although stakeholders from different geographical regions may use different terms, a sound basis is required to allow for reasoning with architectural decisions and proposed alternatives that are stored on the wiki. Elements
described in the core model of architectural knowledge \cite{deBoer_2007} can be useful to implement this structure. \cite{DeBoer_2011} also list emerging research challenges that touch upon knowledge model alignment, knowledge versioning, and knowledge updates. Second, we acknowledge that a wiki populated with architectural knowledge alone does not provide a guarantee to effective architectural knowledge management in global software development; research has shown that the role of the decision-maker is dominant \cite{Al-Ani_2009}. Hence, following common pitfalls regarding the adoption, a clear adoption strategy needs to be chosen, see e.g., \cite{Farenhorst_2008,Mader_2007}. This adoption strategy should first of all lower thresholds of capturing architectural knowledge on the wiki. Criteria for when (and when not) to store architectural knowledge will need to be devised.
The Use of Architectural Knowledge Management Practices in Agile Global Software Development

In this chapter we report on a study conducted to identify what practices for architectural knowledge management in global software development are used in practice. By conducting several interviews with practitioners at three development sites of an organization involved in global software development, we build a thorough view on how the organization uses and manages architectural knowledge across the distributed development sites.

9.1 Introduction

Over the past years, we observed an increase in attention to the management of knowledge in global software development organizations. For these development organizations, timely availability of accurate knowledge is important for effective and efficient software processes (see e.g., (Al-Ani and Redmiles, 2009; Kotlarsky et al., 2008), and the MARK and KNOWING workshop series, held in conjunction with the IEEE Requirements Engineering Conference and IEEE International Conference on Global Software Engineering series respectively).

A specific, yet extremely important kind of knowledge that needs to be shared is knowledge about the software architecture. In a software architecture, the global structure of the system to be built is decided upon. This structure, among others, should capture the major architectural decisions that led to it (Bass et al., 2003). Capturing these architectural decisions facilitates a better decision-making process in shorter time, saving rework and improving the quality of the architecture (Babar et al., 2007; Rus and Lindvall, 2002). Hence, it is important to not only capture the resulting architectural design, but also the decisions, including their rationale, that led to that design: architectural knowledge.

In the GRIFFIN project (GRIFFIN, 2011), we have performed research in the area of architectural knowledge. We define architectural knowledge as the integrated rep-
presentation of the software architecture of a software-intensive system (or a family of systems), the architectural design decisions, and the external context/environment. We have formalized the concept of architectural knowledge in a core model of architectural knowledge (de Boer et al., 2007). Possible reasons for the use of architectural knowledge have been identified and validated as well, see (van der Ven et al., 2006a) and Chapter 2. In the context of GSD, sound management of architectural knowledge can help to overcome the challenges innate to GSD. Architectural knowledge management can be implemented by performing a series of well-defined practices. In the GRIFFIN project, we have developed and initially validated these practices, see e.g., (Clerc, 2008; Clerc et al., 2007a, 2009).

In the research described in this chapter, we have validated the architectural knowledge management practices for their actual use by conducting a survey at a large agile global software development organization. After having interviewed 38 employees across three development sites and having analyzed the results, we conclude that the architectural knowledge management practices that promote decentralization get much more attention than those promoting centralization. We furthermore identified one additional useful practice, ‘peered sites’, covering activities that support a balance in decision-making power across sites.

This chapter is structured as follows. First, §9.2 provides the research question, an overview of the organization where we conducted our survey and details the approach used for this research. Next, §9.3 lists the results of the analysis. In §9.4, we discuss the possible limitations that apply to this research. We conclude this chapter in §9.5 with the conclusions.

9.2 Research Question and Approach

In this section, we first outline our research question (§9.2.1). Next, §9.2.2 provides an overview of the organization where we conducted the survey. In §9.2.3 we describe the approach used for conducting the survey.

9.2.1 Research question

Our primary interest lies in validating the set of practices for architectural knowledge management in global software development. In earlier research, we have developed several practices and validated their usefulness in an organization involved in GSD, see Chapters 3, 6 and 7. Table 9.1 provides an overview of the practices, including references to the earlier research. These practices include frequent interaction across sites, prioritizing a set of rules with which the architecture and underlying decisions should comply, writing down deviations from these architectural rules in case of non-compliance (including a rationale for non-compliance), the urgent request mechanism as a light-weight, non-intrusive manner to quickly gain information on a specific architectural topic of interest, establishing a project structure in which it is clear what the communication responsibilities are, and establishing a single configuration management system in which work products such as documentation, scripts, test cases, and
9.2. Research Question and Approach

source code are stored. For an elaborate description of each individual practice, we refer to the work referenced by Table 9.1 and included in this thesis. The work presented here aims at a broader evaluation of the use of these practices at another large GSD organization.

Hence, our research question is as follows:

What practices for managing architectural knowledge in global software development are used?

Table 9.1: Practices for architectural knowledge management in global software development

<table>
<thead>
<tr>
<th>Reference</th>
<th>Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Frequent interaction across sites</td>
</tr>
<tr>
<td>(Clerc et al., 2007a)</td>
<td>Prioritize architectural rules</td>
</tr>
<tr>
<td>(Clerc et al., 2007a)</td>
<td>Write down deviations from architectural rules</td>
</tr>
<tr>
<td>(Clerc et al., 2007a)</td>
<td>Verify compliance with architectural rules</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Cross-site delegation (designated representatives of each team visit other teams)</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Face-to-face project kick-off</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Urgent request</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Collocated high-level architecture phase</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>Clear organization/project structure (with communication responsibilities)</td>
</tr>
<tr>
<td>(Clerc, 2008; Clerc et al., 2009)</td>
<td>(single) Repository for architecture artifacts</td>
</tr>
<tr>
<td>(Clerc et al., 2009)</td>
<td>Know who is who across the project</td>
</tr>
<tr>
<td>(Clerc et al., 2009)</td>
<td>(single) Configuration management system for work products and source code</td>
</tr>
</tbody>
</table>

9.2.2 Characterization of the organization

We performed the survey at Organization E. Organization E produces high-end printers for the business markets in high-volume printing, wide-format printing and office printing. Organization E is based in the Netherlands and has several additional development sites spread across the globe.

The software in these printers manages user interaction, renders images, and controls the print engine. Software for printers is typically developed by several software development teams, plus architects, integrators, and testers. These software teams reside at the various development sites of the organization.

A typical team structure of project teams at Organization E is depicted in Fig. 9.1. A project manager heads the project and is responsible for its planning, realization, and successful completion. The project manager also agree(s) upon the high-level specifications of the project with upper management and marketing personnel. Requirements and specifications are compiled into product properties by the lead architect in the team. The specifications written by the lead architect start from a user-centric view, i.e., see-
narios on how the end-users will interact with the product. For instance, the architect is responsible to decide what happens in case of an interrupt request from the user of the printer (as in how should it function, which software units should be triggered and how should they function), define the interfaces between these units and the like. Teams also comprise a system integrator who integrates the different software units to build the software system. Additionally, the integrator reports issues encountered during integration and assigns them to the appropriate team or person to be addressed. All three – the project manager, the lead architect, and the integrator – are assigned to a project for its entire duration until the product has been released.

Project teams also include one or more software unit teams that implement the software units. Each unit has a unit leader and a unit architect analogous to the project manager and lead architect at the project level. The unit leader is responsible for planning and organizing. The unit architect transforms the high-level specifications received from the lead architect into detailed technical specifications and passes them to the software developers who implement the code and test it. The unit architects are coordinated by the lead architect, who is often the only team member with the overall view of where (or in which units) the different functionalities of the product reside. Most software units are not developed for a single product but their deliverable is tuned and integrated into several products. A software team can develop a software unit for four, or even more, projects at the same time. This challenges system behavior as well as architecture.

As a basis for our research, we selected two large projects. One of these projects primarily involves the Dutch (NL) site and another site, site A. The other project also involves a third site, site B.

Collaboration between the three sites involved in these two projects differs. In the first project, which involves the NL site and site A, both sites develop distinct software units and integration of the work done by the sites occurs between the software units. In the second project, involving the Dutch site, site A, and site B, collaboration between NL and site B occurs via co-development of the same software unit, whereas collab-
9.2. Research Question and Approach

oration between the Dutch site and site A is organized at the level of unit integration, similar to the first project.

Organization E successfully applies an agile development methodology to encourage creativity and productivity. The organization is flat and workers are encouraged to be proactive in owning up to work responsibilities. Organization E has deliberately opted for cooperation, as opposed to hierarchy, to foster innovation and entrepreneurship.

Agility does not contradict architecture. The agile development process used specifies that the requirement specifications and the architecture design specifications must be approved by the project leaders and the Architecture Council. The lead architect and the unit architects have regular (typically weekly and when needed) meetings to discuss design, and individual teams have their daily meetings.

The agile development methodology that Organization E has been using for the past five years is based on SCRUM [Schwaber and Beedle, 2001]. SCRUM is an agile software development methodology that allows for adaption for use in a GSD setting, see e.g., (Paasivaara et al., 2008; Paasivaara and Lassenius, 2010; Sutherland et al., 2009). A software development cycle at Organization E typically lasts 8 weeks, with sprints of 2 weeks.

9.2.3 Research approach

From the organization, we selected 38 employees working on the two aforementioned projects: 19 were working at the Dutch site, 14 at site A, and 5 at site B. This distribution is proportional to the number of employees at each of the development sites. The roles and experience of the employees ranges from team leaders to designers, architects, and integration responsibles. The majority of these roles has architecting responsibilities.

On average, the employees had 4.45 years of experience in their current role, with a range between 0.5 - 20 years. The employees were working on average 10.33 years at the organization, with a range between 0.5 - 29 years.

We held interviews with the 38 employees individually of about one hour to one-and-a-half hour. The interviews were held according to a predefined questionnaire that was developed in close cooperation with the organization. The topics of the questionnaire covered communication and collaboration, knowledge sharing, relationship (and trust) issues, and quality and productivity. In all these topics, various subquestions were formulated to collect information for the validation of our model; questions regarding communication, collaboration, and knowledge sharing focused on how interaction occurs and how knowledge is shared. We video-recorded the interviews to facilitate a thorough analysis afterwards. In a companion article [Manteli et al., 2011], a subset of these interviews is used to study the impact of governance structures on knowledge management.

We performed content analysis on the coded (transcribed) representations of the text to analyze the interview transcripts and validate our research model. Content analysis is a research method that uses a set of procedures to make valid inferences from text [Weber, 1990]. This helps in e.g., revealing the focus of individuals, or groups. We examined broken down pieces of text or semantic units (not single words, but ‘word sense’) from each interview transcript with our research question in mind and coded an-
answers to the question using an interactive set of concepts; an interactive set of concepts enables us to code what the practitioners actually do, rather than framing it a priori to our validation model. Since we are interested in a possible distinction between architectural knowledge management practices and the sites involved, we code for frequency of the concepts, rather than just their existence (termed ‘conceptual analysis’ by (Weber, 1990)). After we finished coding the 38 interview transcripts, we linked the individual coded fragments to our validation model by inferencing the meaning of the fragments. We performed such a mapping for each of the three sites individually to allow us to compare the results across sites. Intermediate results were discussed in our research team to ensure validity (see §9.4). Fragments that could not be mapped onto the validation model are analyzed separately since these fragments could reveal the existence of other practices in use at the organization that can be leveraged.

9.3 Results

In this section, we describe the results of our study. The validation of the architectural knowledge management practices for global software development is described in §9.3.1. Next, §9.3.2 provides practices that were found to be performed at, or needed by, the organization, but were not part of our validation model.

We have structured the results per development site. Table 9.2 shows the frequency of references made to AKM practices for GSD.

9.3.1 Validation of architectural knowledge management practices for GSD

In this section, we discuss the major findings of Table 9.2. We combine the findings of each of the development sites, yet indicate specific similarities or differences observed. Fig. 9.2 provides an excerpt of the codification and aggregation of concepts to our validation model. In this excerpt, various interaction means (e.g., tools) are shown, and annotated with additional reasons for using these means.

Frequent interaction across sites – The organization uses a wide variety of means to ensure sufficiently frequent interaction across sites: periodic meetings (e.g., weekly or biweekly) to share concerns such as planning issues, bottlenecks, and estimates, the use of video-conferencing and Communicator for face-to-face interaction and desktop/code sharing, and of course email and phone communication. Our analysis shows that especially Communicator-tooling for e.g., video-conferencing is greatly appreciated and meant to lower the barriers of distance in collaboration with remote sites; these findings support the findings from (Damian et al., 2009). However, despite this variety of means, employees indicate that they do not automatically share the result of these local or informal discussions (e.g., coffee-talk) to other sites; rather, the request for that knowledge should be put forward by the other site. This is inherent in the philosophy of the organization: take responsibility, and ask colleagues when needed. We have identified clear differences in the preference to use either the phone (synchronous communication, a
9.3. Results

Table 9.2: Overview of results per site. Numbers between parentheses denote the negation of that practice

<table>
<thead>
<tr>
<th>AKM practice for GSD</th>
<th>Site NL (N = 19)</th>
<th>Site A (N = 14)</th>
<th>Site B (N = 5)</th>
<th>Totals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent interaction across sites</td>
<td>54 (13)</td>
<td>38 (5)</td>
<td>22 (1)</td>
<td>114 (19)</td>
</tr>
<tr>
<td>Prioritize architectural rules</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Write down deviations from architectural rules</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Verify compliance with architectural rules</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Cross-site delegation</td>
<td>23</td>
<td>9</td>
<td>2 (3)</td>
<td>34 (3)</td>
</tr>
<tr>
<td>Face-to-face project kick-off</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Urgent request</td>
<td>13 (1)</td>
<td>1 (1)</td>
<td>–</td>
<td>14 (2)</td>
</tr>
<tr>
<td>Collocated high-level architecture phase</td>
<td>3</td>
<td>–</td>
<td>–</td>
<td>3</td>
</tr>
<tr>
<td>Clear organization/project structure (with comm. responsibilities)</td>
<td>3 (2)</td>
<td>9</td>
<td>2</td>
<td>14 (2)</td>
</tr>
<tr>
<td>(single) Repository for architecture artifacts</td>
<td>1 (5)</td>
<td>1 (4)</td>
<td>(1)</td>
<td>2 (10)</td>
</tr>
<tr>
<td>Know who is who across the project</td>
<td>16</td>
<td>7</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>(single) Configuration management system</td>
<td>1 (36)</td>
<td>3 (8)</td>
<td>1 (3)</td>
<td>5 (47)</td>
</tr>
</tbody>
</table>

Figure 9.2: Excerpt of mapping the content analysis results to a part of our validation model (negations shown in dashed lines, the architectural knowledge management practice ‘frequent interaction across sites’, site B).
higher sense of urgency transferred) or email to share information or ask questions; some people find themselves more confident in using email because it allows them to verify their writings before actually communicating them (i.e., sending the message). These difficulties lie in language barriers (socio-cultural distance) and misinterpretation. At site A, the need for a single whiteboard for distributed design meetings (such as described in [Cataldo et al., 2009]) was expressed. At site B, we observed that Communicator was used heavily and valued very much by the employees; they mentioned that the phone actually was not used that often; in case of an urgent matter, the issue is put forward via email and followed up with a Communicator call to attract attention at the other development site. This links to identified differences in communication styles and preference for selecting communication infrastructure, as reported in the literature and described in [4.3.2].

Prioritize architectural rules, Write down deviations, Verify compliance  – These practices are not used by Organization E. Given the philosophy of entrepreneurship and pro-activity as elaborated on in [9.2.2], Organization E does not pose restrictions such as architectural rules (see Chapter 3) to the employees. Rather, the organization promotes that employees actively request knowledge at the source (i.e., their colleagues) and correct and improve any suboptimal situation they observe along the run.

Cross-site delegation  – The organization heavily relies on cross-site visits to establish trust. Several of the key players (project managers and integration managers) at all sites involved in our study have monthly and bi-monthly visits scheduled in their agendas. Although the interview participants mention that it is necessary to prepare the visits well (to get the most out of them), the general opinion is that the benefits outweigh the drawbacks, as supported by (Sadun, 2010). As an example, an employee from site A indicated:

“And when you are in the mindset where you just say well, the Dutch people are not right then you’re not really building a good relationship. (…) And when you really start to go there and to work with them and find out that there are lot of people that really communicate well and you can make friends there as you can make friends here. But this can only happen if you go there. You cannot make friends by email.”

Gaining trust is particularly important for employees working at site B, the site that was added most recently to the organization. The respondents indicate that cross-site visits explicitly were initiated when site B was started five years ago. Currently, however, the respondents indicate that not all employees from site B are eligible for travelling to the other sites; nor are there any regular team-building events to build trust that include employees working at site B, according to the employees working there.

Face-to-face project kick-off  – Although this practice is not mentioned very frequently by the interviewees, all sites report on having a face-to-face meeting at the start of a project. Apart from that these meetings help to gain trust early in the project (and thus provide a solid basis for future cooperation across sites), employees from the Dutch site indicate that project kick-offs were conducted at site B when this site was started about five years ago.
Urgent request – The organization promotes a low threshold for knowledge sharing and communication; hence, we observe that the Dutch site often poses and receives questions (either via email or Communicator-chat) that quickly need resolution. This shows that the helpfulness and willingness to share knowledge with the other sites (site A and B) is significantly present. A shared goal exists for each of the two projects (delivering correctly functioning software in time), and an urgent request mechanism clearly contributes to that goal by minimizing knowledge sharing delays.

Collocated high-level architecture phase – The practice that promotes the development of the architecture is collocated (i.e., at a centralized location) is not that often mentioned. There is an Architecture Council where decisions that cross-cut projects and affect multiple units are taken. Decisions at the level of an individual unit are left to the unit architect – who may reside at another site –, who has regular meetings with the lead architect of the project.

Clear organization/project structure (with communication responsibilities) – The employees at site A indicate a more hierarchical project organization with clear responsibilities as compared to the Dutch site and site B. As an example, employees from site A indicate that most of the knowledge sharing occurs via (delegation to) the so-called ‘function responsible’ instead of contacting possible knowledge sources directly. Hofstede (2001) shows that this can be attributed to a difference in the degree to which members of organization accept that power is distributed unequally; in other words, a more ‘hierarchical structure’. The organization perceives a change in the difference in hierarchy between the site A and the Dutch site, as indicated by this example:

"Just as an example, when I talk about maybe five, 15, 20 years ago, I was [at site A] and I know people who work in a certain problem; (...) then in a coffee break, I asked a developer about the problem, (...) which was not very much appreciated by my colleague [at site A] (peer) (...) This has changed I think so people are much more lets say equal and its the same flat organization as it is here.”

(single) repository for architecture artifacts – Architectural documents containing architectural decisions and rationale are not stored in a single repository in this organization. Our analysis reveals various examples in which e.g., employees from site B send an e-mail to ask if the information put onto the SharePoint environment is up-to-date. In addition, in the Netherlands, employees ask their colleagues via email to provide them with an architecture document.

Again, these examples show that the organization relies on the pro-activeness of its employees to initiate and promote knowledge sharing across the development sites, and rely little on codified knowledge.

Know who is who across the project – As with the other sites, it is important for the employees at site A to build a trust network with employees at the other sites. Key to building this network is to know ‘who is who’ and to know ‘who knows what’. Means to achieve this include installing a so-called directory (or ‘yellow pages’) as we have observed earlier in (Clerc et al., 2007a) and further elaborated on in (Lago et al., 2010).
We have not observed any such implementation at this organization, yet the goal (to know ‘who is who’) is deemed important.

**Configuration management system** – No shared configuration management system is in place as a knowledge base or for document storage. In the Netherlands, the employees indicate that a myriad of means exist to store information: SharePoint, wikis, a proprietary ‘document finder’ tool, network drives, and email; all are possible locations where information is stored. As a result, artifacts that should be shared (e.g., problem records, meeting minutes) are in fact not shared and employees rely on their colleagues to find out where a certain piece of information is stored:

“There is data management on both sides; some [documents] are shared, some not, and some are duplicated. So sometimes it is difficult to find the right information or the most recently updated documentation.”

With respect to the validation of the architectural knowledge management practices for GSD, we observe that practices that are used most frequently focus on the interaction between employees (by knowing ‘who is who’), using a variety of communication means, travelling, and the possibility to quickly obtain information when needed. On the other hand, practices that focus on capturing and storing knowledge in shared, single repositories or other systems are not used heavily.

### 9.3.2 Additional architectural knowledge management practices

While analyzing the survey results, we found practices for the management of architectural knowledge that were performed or mentioned by the organization, but that were not part of our initial model. In this section, we describe these practices.

At all sites, we found that absence of a policy for strategy for knowledge sharing was mentioned. Moreover, some respondents mentioned that more directing management statements and management support towards knowledge sharing would be beneficial.

In addition, at the Dutch site several employees indicated that a cross-site architecture team (the Architecture Council) is in place, including members from site A. The organization promotes a balance in the amount of and frequency in which knowledge is shared across sites. Nevertheless, some employees indicate that there is a imbalance in that the Dutch site sometimes is dominant in e.g., taking architectural decisions and “(...) having the final say on quality (...);” this imbalance causes difficulties in that employees at other sites feel less involved with the decisions taken. Communication occurs towards other sites, where “(...) [we] hope that they receive it well (...).” These findings are supported by [O’Leary and Mortensen, 2010](#); the configuration of the development sites and the configuration at each development site individually significantly affects the dynamics of the organization. Furthermore, an imbalance in the size of the teams at each site can even invoke competitive and coalitional characteristics [O’Leary and Mortensen, 2010](#).

### 9.3.3 Major findings

In this section, we provide the major findings of our validation.
Organization E performs several activities that lead us to uncover an additional AKM practice for GSD that can be leveraged for organizations involved in GSD: ‘peered sites’. This practice covers the combination of activities that support a balance in decision-making power: establishing a cross-site architecture team, balancing the frequency of information exchange, and setting and promoting a shared goal for the project activities across development sites. In addition, it is suggested to have balanced teams in terms of number of employees working at each site.

In addition, we find that Organization E puts great emphasis on architectural knowledge management practices for GSD that promote decentralization: all sites involved in our study interact frequently with each other without having a pre-defined top-down communication structure. The key players of a site travel to other project locations to share knowledge, employees voluntarily respond to urgent requests for knowledge from their colleagues, and, finally, project kick-offs are held with members from all sites (physically) present. Practices that place a stronger emphasis on centralization, such as establishing a single repository for artifacts or having a single, shared configuration management system are not used; rather, the opposite is true; knowledge and artifacts are stored in multiple locations, causing some difficulties to retrieve the information. Several employees denote a need for having a central repository for artifacts in general and for artifacts that contain architectural knowledge in particular.

The tendency towards decentralization has two major consequences. First, Organization E places a great emphasis on practices that support a personalization approach towards knowledge management as opposed to those that support a codification approach towards knowledge management. This corresponds with the nature in which software development activities are undertaken at this organization: in an agile approach, the pro-activeness and entrepreneurship of individual employees promotes practices that focus on collaboration and bringing the knowledge workers together (see Hansen et al., 1999)), rather than on codifying all knowledge in a single repository. Yet, the employees denote a clear need towards having a repository for artifacts in general and for artifacts that contain architectural knowledge in particular. The myriad of means to store and share knowledge that are in place at the organization result in that several employees do not know where (the latest version of) a document is and whether it is up-to-date or maintained. Following (Desouza and Evaristo, 2004) and (Farenhorst et al., 2007a), a more hybrid knowledge sharing approach combining codification and personalization practices can prove beneficial.

A second consequence of the tendency towards decentralization is that the practices focusing on architectural guidelines that should be adhered to (architectural rules) are not used; posing architectural rules typically occurs in a more centralized collaboration structure. Organization E has deliberately chosen not to pose these rules since the quality mindedness and adherence to quality standards are key responsibilities of each employee individually.
9. The Use of AKM Practices in Agile GSD

9.4 Threats to Validity

In this research, we have validated a collection of architectural knowledge management practices for global software development at a single large organization; we included two projects in our study that involve unit teams spread across three sites. Including only one organization puts some limits on the external validity (generalizability) of our research results. Yet, having included 38 employees with varying tasks and responsibilities at the three sites, we believe to have included a representative subset of a large organization. Furthermore, our results primarily apply to organizations who are involved in agile-based global software development. Organizations that support a more hierarchical and/or formal software development methodology might benefit from practices that are proposed but are not used by the organization where we conducted our survey.

We have used content analysis as the basis of our study. As indicated by (Weber, 1990), several difficulties exist in the application of content analysis that may pose validity issues. We discuss these briefly in the remainder of this section. First, with respect to measurement, we decided to count the number of occurrences of the semantic unit. Weber (1990) argues that in counting semantic units and subsequently grouping them to an element of our validation model, it requires less effort to identify successive mentions of a word as compared to the first occurrence; however, consistent use in terminology (i.e., Organization E’s terminology) has eased this process. Furthermore, we have also investigated possible reasons for not being able to link any semantic unit to our validation model (see §9.3.2). Second, with respect to indication, we have clearly added no or non in a semantic unit to indicate the negation of a practice or element (see Fig. 9.2). Since we are interested to use the interview transcripts to answer our research questions and not compare interviews or interviewees individually, we have not paid much attention to the use of adjectives or other semantical information (such as understated or overstated words). Third, with respect to representation and interpretation, we have performed the content analysis of the 38 interviews in a phased approach. This approach allowed us to reflect on possible ambiguities in terms used within our research group.

9.5 Conclusions

We have conducted a survey at a large agile multi-site organization (Organization E) involved in global software development to identify how architectural knowledge actually is shared across development sites. We have analyzed the results of 38 interviews held with employees from three development sites active in two large projects of the organization. Analysis of the results and relating them to our validation model helps to identify the pros and cons of using architectural knowledge and architectural knowledge management practices.

We conclude that architectural knowledge management practices that promote decentralization get much more attention than those promoting centralization at the agile GSD organization. Organization E uses practices that focus on interaction and knowledge sharing bottom-up, e.g., between employees individually instead of putting for-
ward (centralized) guidelines and tools. This focus on decentralization leads to an em-
phasis by Organization E on personalization practices (e.g., bringing knowledge work-
ers together) as compared to codification practices.

Finally, we identified one additional useful practice for architectural knowledge
management in GSD: ‘peered sites’. This practice covers activities that support a bal-
ance in decision-making power across sites.
Conclusions

In this thesis, we have studied how organizations involved in global software development perform architectural knowledge management activities. We have studied several organizations and developed and validated a series of architectural knowledge management practices to help overcome challenges involved GSD. In this chapter, we revisit the research questions posed in Chapter 1 and summarize the answers provided by our research. Furthermore, we provide a broader perspective on the results or our research and point out directions for further research based upon this work.

10.1 Contributions

In this section, we revisit the research questions answered in this thesis and summarize the answers our research provided.

In this thesis we have studied how architectural knowledge can be managed in organizations that are involved in global software development. As shown in Fig. 1.2 in Chapter 1 we have elaborated this central research question using a multi-disciplinary approach leading to four more detailed research questions. In this section, we first answer these detailed research questions in §10.1.1 through §10.1.4. Next, in §10.1.5 we revisit our central research question.

10.1.1 How is architectural knowledge used? (RQ1)

Recent advances in the discipline of software architecture strengthen the notion of architectural knowledge as pivotal to prevent design erosion and reduce maintenance costs. In Chapter 2 we report on a study that is aimed at identifying how architectural knowledge in fact is used in the field. By collecting Dutch practitioners’ views on the use of so-called use cases for architectural knowledge and combining their responses, we identified five main clusters that describe the use of architectural knowledge: ‘use as an architectural decision set’, ‘use for taking decisions to address the problems or concerns of stakeholders’, ‘use for tracing decisions from requirements, via design, to the implementation’, ‘use for performing risk or tradeoff analyses’, and ‘use for stakeholder communication’. We used the responses of the practitioners and constructed the mindset of the architect, relative to these five clusters.
Our study answers RQ[1] in that it shows that the mindset of architects is focused on delivering a solution and capturing the related architectural decisions. Consequently, we conjecture that a so-called micro view on software architecture largely is in place: architects are focused on developing an architecture for a specific solution and (more and more) on capturing the architectural decisions and rationale for that solution. In addition, the architectural knowledge is used for communication with stakeholders on the progress of the solution. What lacks in the mindset of architects is a view that exceeds specific architectures but puts architectures in context by validating the architectures and the architectural decisions that led to them by means of analyses and assessments.

10.1.2 What are practices for managing architectural knowledge in a global software development environment? (RQ[2])

In this thesis, we report on several studies that aim to identify practices for architectural knowledge management in global software development, and hence provide an answer to RQ[2]. We have identified the approaches that typical GSD organizations adhere to in ensuring compliance across development sites. In Chapters [3] and [4], we show that practices related to the development process can suffice (at Organization B) and are needed (following the suggestions made at Organization A) to further secure compliance with architectural rules. These practices pertain to communication of the architectural knowledge across sites, to manage deviations of architectural decisions (i.e., manage non-compliance), and to explicitly perform compliance verification activities.

In Chapter [5], we learned, by studying software product assessments performed at Organization C, that the quality of software products developed using GSD is not further increased by capturing architectural knowledge in the software products. Hence, we have not found any reasons that the product counterpart of the aforementioned process practices for architectural knowledge management is important in GSD.

Building upon the insight gained thusfar, we performed an extensive literature study in the discipline of requirements engineering to identify practices that could prove useful in the related field of software architecture. In Chapter [6], we report the resulting practices. The seven practices identified mainly support a personalization strategy towards knowledge management; frequent interaction and communication between practitioners involved in the software architecture that reside at different development sites is advocated. Only one of the practices purely supports a codification strategy towards knowledge management; codifying knowledge in a repository for architecture artifacts. The practices are described in detail in §[6.3] starting on page [86].

We used the practices that were distilled from the literature, augmented the practices with some organization-specific experiences and practices, and determined the perceived usefulness of the combined set of practices at Organization D. This case study, reported in Chapter [7], provides us with confidence that the practices distilled are regarded as useful. Several practices (e.g., traveling to other project locations) are deemed particularly useful since this study showed that software development projects that evolve towards using three sites experience a critical need to plan for (the implementation of) these architectural knowledge management practices in advance.
10.1. Contributions

10.1.3 How can architectural knowledge management (practices) in global software development be supported (by tools)? (RQ-3)

The use of architectural knowledge in global software development can further be strengthened by providing practitioners with appropriate tools. For answering RQ-3 we have performed a study (reported in Chapter 3) on the use of wikis and semantic wikis to implement practices for architectural knowledge management in GSD.

First, we identified to what extent the architectural knowledge management practices for GSD may be implemented using wikis. We related the practices to generic wiki functionalities and conclude that wikis form a good mechanism to implement a hybrid strategy for managing architectural knowledge in GSD. Furthermore, we concluded that a substantial part of the AKM practices may be implemented using wikis.

Second, we implemented three use cases for architectural knowledge in a semantic wiki. The use cases we implemented support the architect in reasoning with architectural knowledge, independent of location, time zone, or socio-cultural borders. Once the architectural knowledge is captured in the semantic wiki, it may help in achieving the benefits of GSD.

10.1.4 Which practices for managing architectural knowledge are used in a global software development environment? (RQ-4)

During our research, we have obtained a good overview of the major building blocks that constitute architectural knowledge. We performed a case study at Organization E to identify how architectural knowledge in fact is used in a typical global software development organization by validating the use of the architectural knowledge management practices for GSD. This helps us to answer RQ-4.

The results of the study show that architectural knowledge management practices that promote decentralization get much more attention than those promoting centralization at Organization E. Consequently, the practices that support a personalization strategy towards knowledge management are more often used than those that support a codification strategy. Furthermore, we identified one additional useful practice for AKM in GSD: “peered sites”. This practice covers activities that support a balance in decision-making power across sites.

By comparing these results with the previous studies reported on in this thesis, we reassess the usefulness of the ‘peered sites’ practice by pointing to several symptoms observed at the other case study organizations: the architectural knowledge management practice ‘cross-site delegation’ was deemed useful at Organization D since it was not explicitly planned for in projects that evolved to using three sites. Also, we observed that the various sites of Organization B regard each other as peers, setting a shared goal for the project activities, which contributes to shared team understanding.
10. Conclusions

10.1.5 How can architectural knowledge be managed in a global software development environment?

In this thesis we have identified and validated a series of practices for architectural knowledge management in global software development. The practices primarily focus on measures in the process to ensure smooth cooperation and collaboration between various stakeholders across multiple development sites involved in architecting activities. A minority of the architectural knowledge management practices actually focuses on the infrastructure that is recommended to be present in GSD organizations: ‘having a shared configuration management system’ and ‘having a single repository for architecture artifacts’. These practices allow for adequate codification of architectural knowledge the practices that subsequently be shared using other architectural knowledge management practices.

The architectural knowledge management practices we contribute are aimed at overcoming distance involved in GSD: distance in the various ways in which it manifests itself in GSD. The physical distance can be overcome because most, if not all, of the practices can be at least partly supported by tooling such as instant messaging, wikis, and other collaboration tooling. Other practices, focusing on frequent interaction across sites (face to face project kickoff, cross-site delegation, and a collocated high-level architecture phase) actually bridge the physical distance and time difference between the development sites to address the socio-cultural distance and to build trust. When a situation is reached in which the sites involved in a GSD project perceive each other and act as “peers”, we can conclude that all forms of distance are overcome in at least a sufficient way so that the GSD challenges are addressed and the full benefits of GSD can be reaped.

10.2 Discussion

In this section, we provide a discussion of the results of this thesis. In §10.2.1 we provide directions to other fields of interest that are related to the research performed. In §10.2.2 we reflect on the contributions as described in this thesis. Finally, in §10.2.3 we provide a discussion on several advances in the field that may be further fueled by our research and call for additional research efforts.

10.2.1 Linking to other adjacent fields

When put in a broader perspective, the results of the research presented in this thesis relate to the concept of boundary spanning, as identified in the information systems’ literature (Tushman, 1977; Allen, 1977; Nochur and Allen, 1992; Levina and Vaast, 2005a). Boundary spanning is emphasized by the literature as relying on individuals (i.e., the boundary spanners) to facilitate the sharing of expertise by linking two or more groups of people separated by location, hierarchy, or function (Cross and Parker, 2004). Interestingly, the literature emphasized the importance of designating boundary spanning roles, such as “representative” versus “gatekeeper” and “advice” versus “trust broker”.

142
10.2. Discussion

Our research supports and further details the views on boundary spanning put forward by Levina and Vaast in that we provide a series of practices that either focus on the boundary spanner or on the boundary object – boundary spanners are individuals that facilitate the sharing of expertise by linking two or more groups of people separated by location, hierarchy, or function (Cross and Parker, 2004); a boundary object is the object through which (codified) knowledge is exchanged (Wenger, 1998), e.g., via organizations’ intranets or repositories. In our research, we have shown that global software development organizations involved in the management of architectural knowledge tend to place more emphasis on personalization strategies, emphasizing the importance of the boundary spanner, rather than codification strategies, emphasizing the importance of the boundary object. Nevertheless, as pointed out by (Levina and Vaast, 2005a), it is not only important to appoint a boundary spanner but also to monitor how the practitioners enact their roles. This notion is reflected in the emergence of agile GSD which is not so much focused on designated hierarchy in terms of (reporting) responsibilities, but rather focuses on the actual collaboration – aimed at uncovering (hidden, tacit) knowledge necessary to overcome GSD challenges and speed up software development.

Oshri et al. (2008) define a series of practices or mechanisms that show close resemblance to the work described in this thesis: having personalized directories, standardized templates or work processes, frequent teleconferencing sessions, and occasional short visits all contribute to the notion of ‘who knows what’. The results of Oshri et al. are directly supported by our research, including the focus on personalization practices and the results found at e.g., Organization E. Kotlarsky et al. (2009), finally, indicate that activating a transactive memory system helps in bridging knowledge boundaries across geographic borders. Furthermore, Kotlarsky et al. show that a transactive memory system relies to a greater extent on personalized directories than on codified directories – personalized directories mainly residing in people’s heads.

10.2.2 Reflections on our contributions

One may object to the results presented in this thesis that they mainly focus on sound collaboration and are not really architecture-specific. Mainly, the architectural knowledge management practices that support a personalization strategy towards knowledge management could be referred to as common sense, whereas the practices that support a codification strategy and e.g., the architectural rules cover the real ‘architectural part’. Perhaps, this is true. Our research, however, shows that better architectural knowledge management can be achieved by focusing on process and personalization practices: the practices are deemed useful especially because they aren’t used but needed. Process practices that are in use prove valuable (see e.g., the results observed at Organization D and Organization E) and in fact can be easily supported by tools that support collaboration across borders. In other words, not the availability of shared architecture tooling is important, but rather tooling that is focused on collaboration and communication across development sites.

In other words, the main difference appears to exist not so much in the details of the architectural knowledge itself but rather in the process by means of which the knowledge is managed across development sites. Surely, the process practices concern archi-
10. Conclusions

Our results require the architect to fully embrace the architectural knowledge management practices and behave and act accordingly. As Booch (2011) puts it, although “The best architecture is (...) one that structures the technical core (...)”, he finds that “However, the social elements of growing software-intensive systems are much, much messier”.

10.2.3 A glance at the future

This research aims to unite the disciplines of architectural knowledge management and global software development. As this thesis has shown, no single project management or software development method is presupposed in order to address challenges in GSD by sound management of architectural knowledge. Rather, our research points to several practices that can be applied in certain contexts to overcome the challenges. Several literature sources (see e.g., §1.2.7) report that a greater emphasis on documentation (codification) and tooling that support a codification strategy towards knowledge management is necessary when software development organizations are involved in GSD. Our studies, however, support the urge to use codification practices only to a limited extent; more emphasis is needed for personalization practices as compared to codification practices. We acknowledge that the context in which the practices are applied, may differ. In our research, we primarily validated the use of the architectural knowledge management practices in an organization involved in agile GSD. A change of context does require a reassessment of the applicability of the practices, as was discussed during the recent IFIP WG 2.10 meeting. We see two future developments linked to these research results on that are of interest.

First, although we show that personalization practices can be used to share knowledge, one should ask himself whether this scales perpetually. Recall Parnas’ remark in the sidebar in Agerfalk and Fitzgerald (2006); when turning to GSD, choosing the right order in which to implement more codification-oriented practices and personalization-oriented practices is important. We support ideas for developing a pattern-like cookbook that, based on organizational, cultural, and possibly technological factors that are currently not fully understood, may help in deciding what architectural knowledge management practice to apply when. We are led to believe that our list of practices and the insights obtained during this study can serve as a useful starting point.

Second, one may regard the role of professionals and individuals to become even of greater importance; not so much to retain or capture the technical knowledge about the architecture itself, but rather to show the actual behaviour, willingness, and competences to stand out as an architectin effectively sharing and managing architectural knowledge across time zones and borders. As these competences are becoming increasingly important, future research may be dedicated to identify the DNA of a true global software architect.
Appendix
Overview of the Perceived Usefulness of AKM Practices

This appendix lists the results of study to the perceived usefulness of the architectural knowledge management practices, as described in Chapter 7. For each practice, we indicate the number of respondents that indicated their perceived usefulness (N) and Spearman’s rho (ρ) as a measure of the correlation between the practice and the number of sites and the perceived usefulness of the practice, including its significance (p-value).

Figure A.1: Frequent interaction across sites \( N = 85 \), \( \rho = -0.058 \), p-value = 0.599
Overview of the Perceived Usefulness of AKM Practices

Figure A.2: Cross-site delegation (representatives of each team visit other teams) $N = 88$, $\rho = -0.004$, $p$-value = 0.973

Figure A.3: Face-to-face project kick-off meeting $N = 88$, $\rho = 0.093$, $p$-value = 0.389

Figure A.4: Urgent request (email, mailing list or telephone to quickly get information) $N = 88$, $\rho = -0.015$, $p$-value = 0.886
Overview of the Perceived Usefulness of AKM Practices

Figure A.5: Have a clear organization/project structure with communication responsibilities $N = 88$, $\rho = -0.125$, $p-value = 0.245$

Figure A.6: Travel to other project locations $N = 88$, $\rho = 0.154$, $p-value = 0.151$

Figure A.7: Know who is who across the project (directory) $N = 88$, $\rho = -0.041$, $p-value = 0.702$
Overview of the Perceived Usefulness of AKM Practices

Figure A.8: Have a single repository for architecture artifacts (establish a repository for architecture artifacts) \( N = 88, \rho = -0.088, p\text{-value} = 0.415 \)

Figure A.9: Have a shared infrastructure for work products and source code (configuration management) \( N = 88, \rho = 0.019, p\text{-value} = 0.860 \)
Samenvatting

Het beheren van architectuurkennis in wereldwijde softwareontwikkeling

Het vakgebied dat zich bezighoudt met het op een beheerste wijze ontwikkelen van software wordt ook wel ‘software engineering’ genoemd. Twee recente ontwikkelingen in dit vakgebied betreffen de begrippen ‘architectuurkennis’ en ‘global software development’. Onder ‘architectuurkennis’ verstaan we de geïntegreerde representatie van de architectuur van een systeem of een familie van systemen, de ontwerpbeslissingen en de externe context. Het beheren van architectuurkennis leidt tot meer inzicht in de afwegingen van de softwarearchitect, voorkomt het optreden van ‘design-erosie’ en vergemakkelijkt het doorvoeren van wijzigingen aan het systeem. Wereldwijde softwareontwikkeling oftewel ‘global software development’ (GSD) betreft de ontwikkeling van software waarbij teams vanuit geografisch verspreide locaties samenwerken. Wanneer we dit vergelijken met softwareontwikkeling die op één locatie plaatsvindt, leidt GSD tot enkele additionele uitdagingen. Deze uitdagingen vinden hun oorsprong in de afstand: de fysieke afstand, de afstand in tijd (tijdsverschil) en de socio-culturele afstand (cultuur en taal).

Binnen het GRIFFIN-onderzoeksproject, waar dit proefschrift een resultaat van is, hebben industriële en academische partners onderzoek verricht naar betere hulpmiddelen en ondersteuning voor het beheren van architectuurkennis. In het onderzoeksproject zijn diverse perspectieven onderkend, zoals het delen van architectuurkennis binnen en tussen organisaties, het ontdekken van relevante architectuurkennis in bestaande systemen en documentatie, het traceren van de relaties en verbanden tussen ontwerpbeslissingen en andere onderdelen van architectuurkennis en, ten slotte, het delen en beheren van architectuurkennis in een GSD-situatie met daarin specifieke aandacht voor het perspectief van compliance. Dit proefschrift richt zich op het onderzoek naar dit laatste perspectief en geeft antwoord op de vraag hoe architectuurkennis beheerd kan worden in een GSD-situatie.

Ons onderzoek baseert zich op diverse studies die bij verschillende industriële partners zijn uitgevoerd. Om inzicht te verkrijgen in de wijze waarop architectuurkennis in de praktijk wordt gebruikt, hebben wij een initiële studie uitgevoerd onder de Nederlandse IT-architectengemeenschap zoals deze werkzaam is bij diverse toonaangevende IT-organisaties, -afdelingen, of -platformgroepen in Nederland. We hebben inzicht verkregen in de mogelijke redenen waarom IT-architecten architectuurkennis gebruiken. Deze studie heeft aangetoond dat IT-architecten voornamelijk architectuurkennis gebruiken om gericht een oplossing voor een ontwerpprobleem (het te ontwerpen systeem) na te streven. Architectuurkennis wordt, volgens de studie, in beperkte mate gezien als een verzameling ontwerpbeslissingen waarin discrepanties, inconsistenties of openstaande issues kunnen worden herkend. Evenmin gebruiken de IT-architecten de kennis voor het valideren van de architectuur en de ontwerpbeslissingen.

Een tweede studie is uitgevoerd bij een organisatie die op vier verschillende, geografisch verspreide locaties software ontwikkelt en waar een centraal gesitueerd team architectuurrichtlijnen - een specifieke vorm van architectuurbeslissingen - uitvaardigt
Het beheren van architectuurkennis in wereldwijde softwareontwikkeling

voor ontwikkelteams op andere locaties. Deze studie resulteerde in procesmatige aanbevelingen om de architectuurregels beter te laten bekijken bij de ontvangende teams. Een vervolgstudie, waarin wij deze organisatie hebben vergeleken met een andere organisatie, bevestigde ons inzicht dat met name procesmaatregelen uitkomst bieden: active communicatie van architectuurrichtlijnen over verschillende locaties, het registreren van afwijkingen van de richtlijnen en expliciete verificatie van conformiteit aan deze richtlijnen.

Aangezien het beheren van architectuurkennis niet slechts door oplossingen die procesmatig van aard zijn ondersteund hoeft te worden, richtte een volgende studie zich op de vraag of er ook in de ontwikkelde producten zoals broncode en (systeem-)documentatie architectuurkennis kan worden vastgelegd. Hiertoe zijn acht uitgevoerde productbeoordelingen in retrospectief bekeken om vast te stellen hoe architectuurkennis in het product wordt vastgelegd. Voorts is onderzocht of er verschillen bestaan in het vastleggen van architectuurkennis in producten die op één enkele site ontwikkeld zijn versus producten die met GSD zijn ontwikkeld. Dit onderzoek gaf ons het inzicht dat tussen beide categorieën producten geen verschil bestaat in de mate waarin architectuurkennis in het product wordt vastgelegd.

Om toe te werken naar hulpmiddelen voor het beheren van architectuurkennis in GSD zijn enkele praktijken opgesteld door een vergelijking met de discipline van ‘requirements engineering’ (eisenontwikkeling). Deze discipline kent gelijkenissen met die van het beheren van architectuurkennis en in de literatuur zijn reeds diverse oplossingen voor requirements engineering in GSD-situaties bekend. De oplossingen die eveneens geschikt zijn voor het beheren van architectuurkennis zijn vertaald in praktijken. De praktijken richten zich op frequente interactie tussen verschillende locaties, het bezoeken van andere teams door aangewezen verantwoordelijken, een gezamenlijke projectstartbijeenkomst, een mechanisme (telefoon, e-mail, mailinglijst) om snel architectuurkennis te verkrijgen, een duidelijke (project-)organisatie met belegde verantwoordelijkheden voor communicatie, het reizen naar andere locaties, kennis van ‘wie is wie’ binnen het ontwikkelproject, één enkele verzameling waar architectuurartefacten worden vastgelegd en, ten slotte, een gezamenlijke infrastructuur voor het vastleggen van werkproducten en broncode. Van de opgestelde praktijken is in een case study het gepercipieerde nut bepaald van deze praktijken specifiek voor projecten die op meerder locaties ontwikkeld zijn. In het algemeen worden de praktijken voor het beheren van architectuurkennis als nuttig ervaren; tevens blijkt uit dit onderzoek dat het nut zeer groot is voor projecten die verspreid over drie locaties worden uitgevoerd, hetgeen duidt op een duidelijke behoefte voor de implementatie van deze praktijken. Een deel van ons onderzoek naar praktijken voor het beheren van architectuurkennis in GSD-situaties richt zich op het identificeren en ontwikkelen van adequate geautomatiseerde hulpmiddelen.

Tijdens onze laatste, afsluitende studie is onderzocht in hoeverre de praktijken voor het beheren van architectuurkennis ook daadwerkelijk gebruikt worden in projecten die met behulp van wendbare (‘agile’) ontwikkelmethoden en GSD worden gerealiseerd. Uit dit onderzoek blijkt dat praktijken die zich richten op het samenbrengen van de medewerkers van de organisatie meer gebruikt worden dan praktijken die centralisatie
Het beheren van architectuurkennis in wereldwijde softwareontwikkeling

nastreven door (centraal) architectuurkennis vast te leggen. Voorts blijkt uit dit onderzoek dat het voordelen biedt om gelijkenis tussen sites te bewerkstelligen, zoals het hebben van een architectuurteam met leden van verschillende locaties, het uitdragen van een gezamenlijk projectdoel en het verdelen van de beslissingsmacht.

De bijdrage van dit proefschrift betreft, samenvattend, een gevalideerde verzameling van praktijken om het beheren van architectuurkennis en diverse inzichten over de daadwerkelijke of aanbevolen toepassing van deze praktijken.
SIKS Dissertation Series

1998-01 Johan van den Akker (CWI)
   DEGAS - An Active, Temporal Database of Autonomous Objects

1998-02 Floris Wiesman (UM)
   Information Retrieval by Graphically Browsing Meta-Information

1998-03 Ans Steuten (TUD)
   A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-04 Dennis Breuker (UM)
   Memory versus Search in Games

1998-05 E.W. Oskamp (RUL)
   Computerondersteuning bij Straftoemeting

1999-01 Mark Sloof (VU)
   Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-02 Rob Potharst (EUR)
   Classification using decision trees and neural nets

1999-03 Don Beal (UM)
   The Nature of Minimax Search

1999-04 Jacques Penders (UM)
   The practical Art of Moving Physical Objects

1999-05 Aldo de Moor (KUB)
   Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-06 Niek J.E. Wijngaards (VU)
   Re-design of compositional systems

1999-07 David Spelt (UT)
   Verification support for object database design

1999-08 Jacques H.J. Lenting (UM)
   Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000-01 Frank Niessink (VU)
   Perspectives on Improving Software Maintenance

2000-02 Koen Holtman (TUE)
   Prototyping of CMS Storage Management

2000-03 Carolien M.T. Metselaar (UvA)
   Sociaal-organisatorische gevolgen van kennisstechnologie; een procesbenadering en actorenperspectief

2000-04 Geert de Haan (VU)
   ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-05 Ruud van der Pol (UM)
   Knowledge-based Query Formulation in Information Retrieval

2000-06 Rogier van Eijk (UU)
   Programming Languages for Agent Communication

2000-07 Niels Peek (UU)
   Decision-theoretic Planning of Clinical Patient Management

2000-08 Veele Coup (EUR)
   Sensitivity Analysis of Decision-Theoretic Networks

2000-09 Florian Waas (CWI)
   Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
   Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
   Scalable Distributed Data Structures for Database Management

2001-01 Silja Renooij (UU)
   Qualitative Approaches to Quantifying Probabilistic Networks

2001-02 Koen Hindriks (UU)
   Agent Programming Languages: Programming with Mental Models

2001-03 Maarten van Someren (UvA)
   Learning as problem solving

2001-04 Evgeni Smirnov (UM)
   Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-05 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-06 Martijn van Welie (VU)
Task-based User Interface Design

2001-07 Bastiaan Schönhaege (VU)
Diva: Architectural Perspectives on Information Visualization

2001-08 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics

2001-09 Pieter Jan ’t Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Stierhuis (UvA)
Modeling and Simulating Work Practice
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VU)
Knowledge Management:
The Role of Mental Models in Business Systems Design

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (UvA)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UvA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
2003-07 Machiel Jansen (UvA)  
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)  
Repair Based Scheduling

2003-09 Rens Kortmann (UM)  
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)  
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)  
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordeman (UT)  
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)  
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)  
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerd (TUD)  
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)  
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)  
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)  
Learning Search Decisions

2004-01 Virginia Dignum (UU)  
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)  
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)  
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UvA)  
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)  
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)  
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)  
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)  
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)  
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UvA)  
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)  
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)  
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)  
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)  
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)  
Multi-Relational Data Mining

2004-16 Federico Divina (VU)  
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)  
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)  
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20
Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005-01
Floor Verdenius (UvA)
Methodological Aspects of Designing Induction-Based Applications

2005-02
Erik van der Werf (UM)
AI techniques for the game of Go

2005-03
Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04
Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05
Gabriel Infante-Lopez (UvA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06
Pieter Spronck (UM)
Adaptive Game AI

2005-07
Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08
Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09
Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10
Anders Bouwer (UvA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11
Eith Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12
Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13
Fred Hamburg (UL)
Een Computermodell voor het Ondersteunen van Euthanasiebeslissingen

2005-14
Borys Omelanyenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15
Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16
Joris Graaumans (UU)
Usability of XML Query Languages

2005-17
Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18
Danielle Sent (TUD)
Test-selection strategies for probabilistic networks

2005-19
Michel van Dartel (UM)
Situated Representation

2005-20
Cristina Coteam (UL)
Cyber Consumer Law. State of the Art and Perspectives

2005-21
Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006-01
Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02
Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03
Noor Christoph (UvA)
The role of metacognitive skills in learning to solve problems

2006-04
Marta Sabou (VU)
Building Web Service Ontologies

2006-05
Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06
Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07
Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08
Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09
Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivity - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkin (UvA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlović (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UvA)
Focused Information Access using XML Element Retrieval

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy; a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UvA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Hub Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
SIKS Dissertation Series

Meetings in Smart Environments: Implications of Progressing Technology
Nick Bergboer (UM) 2007-14

Context-Based Image Analysis
Joyca Lacroix (UM) 2007-15

NIM: a Situated Computational Memory Model
Davide Grossi (UU) 2007-16

Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
Theodore Charitos (UU) 2007-17

Reasoning with Dynamic Networks in Practice
Bart Orriens (UvT) 2007-18

On the development and management of adaptive business collaborations
David Levy (UM) 2007-19

Intimate relationships with artificial partners
Slinger Jansen (UU) 2007-20

Customer Configuration Updating in a Software Supply Network
Karianne Vermaas (UU) 2007-21

Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
Zlatko Zlatev (UT) 2007-22

Goal-oriented design of value and process models from patterns
Peter Barna (TUE) 2007-23

Specification of Application Logic in Web Information Systems
Georgina Ramirez Camps (CWI) 2007-24

Structural Features in XML Retrieval
Joost Schalken (VU) 2007-25

Empirical Investigations in Software Process Improvement
Joost Schalken (VU) 2007-26

Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
Katalin Boer-Sorbn (EUR) 2008-01

On Computer-Aided Methods for Modeling and Analysis of Organizations
Alexei Sharpskykh (VU) 2008-02

Optimizing hierarchical menus: a usage-based approach
Vera Hollink (UvA) 2008-03

Management of Uncertain Data - towards unattended integration
Andr de Keijzer (UT) 2008-04

Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
Bela Mutschler (UT) 2008-05

On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
Arjen Hommersom (RUN) 2008-06

Supporting the tutor in the design and support of adaptive e-learning
Janneke Bolt (UU) 2008-07

Bayesian Networks: Aspects of Approximate Inference
Christof van Nimwegen (UU) 2008-08

The paradox of the guided user: assistance can be counter-effective
Wauter Bosma (UT) 2008-09

Discourse oriented summarization
Vera Kartseva (VU) 2008-10

Designing Controls for Network Organizations: A Value-Based Approach
Jozsef Farkas (RUN) 2008-11

A Semiotically Oriented Cognitive Model of Knowledge Representation
Caterina Carracolo (UvA) 2008-12

Topic Driven Access to Scientific Handbooks
Arthur van Bunningen (UT) 2008-13

Context-Aware Querying: Better Answers with Less Effort
Henriette van Vugt (VU) 2008-14

The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains
Martijn van Otterlo (UT) 2008-15

Embodied agents from a user’s perspective
Henning Rode (UT) 2008-16

Applying Architecture and Ontology to the Splitting and Allying of Enterprises
Guido de Croon (UM) 2008-17

Adaptive Active Vision
Rex Arendsen (UvA) 2008-18
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UvA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huibregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogen (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UvA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukunya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietske Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UvA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UvA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuth (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotiuky (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UvA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
| 2009-17 | Laurens van der Maaten (UvT) | Feature Extraction from Visual Data |
| 2009-18 | Fabian Groffen (CWI) | Armada, An Evolving Database System |
| 2009-19 | Valentin Robu (CWI) | Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets |
| 2009-20 | Bob van der Vecht (UU) | Adjustable Autonomy: Controlling Influences on Decision Making |
| 2009-21 | Stijn Vanderlooy (UM) | Ranking and Reliable Classification |
| 2009-22 | Pavel Serdyukov (UT) | Search For Expertise: Going beyond direct evidence |
| 2009-23 | Peter Hofgesang (VU) | Modelling Web Usage in a Changing Environment |
| 2009-24 | Annerieke Heuvelink (VUA) | Cognitive Models for Training Simulations |
| 2009-25 | Alex van Ballegooij (CWI) | “RAM: Array Database Management through Relational Mapping” |
| 2009-26 | Fernando Koch (UU) | An Agent-Based Model for the Development of Intelligent Mobile Services |
| 2009-27 | Christian Glahn (OU) | Contextual Support of social Engagement and Reflection on the Web |
| 2009-28 | Sander Evers (UT) | Sensor Data Management with Probabilistic Models |
| 2009-29 | Stanislav Pokraev (UT) | Model-Driven Semantic Integration of Service-Oriented Applications |
| 2009-30 | Marcin Zukowski (CWI) | Balancing vectorized query execution with bandwidth-optimized storage |
| 2009-31 | Sofiya Katrenko (UvA) | A Closer Look at Learning Relations from Text |
| 2009-32 | Rik Farenhorst (VU) and Remco de Boer (VU) | Architectural Knowledge Management: Supporting Architects and Auditors |
| 2009-33 | Khiet Truong (UT) | How Does Real Affect Affect Affect Recognition In Speech? |
| 2009-34 | Inge van de Weerd (UU) | Advancing in Software Product Management: An Incremental Method Engineering Approach |
| 2009-35 | Wouter Koelwijl (UL) | Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling |
| 2009-36 | Marco Kalz (OUN) | Placement Support for Learners in Learning Networks |
| 2009-37 | Hendrik Drachsler (OUN) | Navigation Support for Learners in Informal Learning Networks |
| 2009-38 | Riina Vuorikari (OU) | Tags and self-organisation: a metadata ecology for learning resources in a multilingual context |
| 2009-40 | Stephan Raaijmakers (UvT) | Multimomial Language Learning: Investigations into the Geometry of Language |
| 2009-41 | Igor Berezhnyy (UvT) | Digital Analysis of Paintings |
| 2009-42 | Toine Bogers | Recommender Systems for Social Bookmarking |
| 2009-43 | Virginia Nunes Leal Franqueira (UT) | Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients |
| 2009-44 | Roberto Santana Tapia (UT) | Assessing Business-IT Alignment in Networked Organizations |
| 2009-45 | Jilles Vreeken (UU) | Making Pattern Mining Useful |
| 2009-46 | Loredana Afanasiev (UvA) | Querying XML: Benchmarks and Recursion |
| 2010-01 | Matthijs van Leeuwen (UU) | Patterns that Matter |
| 2010-02 | Ingo Wassink (UT) | Work flows in Life Science |
| 2010-03 | Joost Geurts (CWI) | |
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software.
Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriëtte Cramer (UvA)
People’s Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steunenbrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov (TUD)
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic relational patterns
2010-29 Stratos Idreos (CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT)
Accessing natural history - discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UvA)
Ontology enrichment from heterogeneous sources on the Web
2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT)
Modeling representation uncertainty in concept-based multimedia retrieval
2010-34 Teduh Dirgahayu (UT)
Interaction design in service compositions
2010-35 Dolf Trieschnigg (UT)
Proof of concept: Concept-based biomedical information retrieval
<table>
<thead>
<tr>
<th>Year</th>
<th>Author</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010-36</td>
<td>José Janssen (OU)</td>
<td>Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification</td>
</tr>
<tr>
<td>2010-37</td>
<td>Niels Lohmann (TUE)</td>
<td>Correctness of services and their composition</td>
</tr>
<tr>
<td>2010-38</td>
<td>Dirk Fahland (TUE)</td>
<td>From Scenarios to components</td>
</tr>
<tr>
<td>2010-39</td>
<td>Ghazanfar Farooq Siddiqui (VU)</td>
<td>Integrative modeling of emotions in virtual agents</td>
</tr>
<tr>
<td>2010-40</td>
<td>Mark van Assem (VU)</td>
<td>Converting and Integrating Vocabularies for the Semantic Web</td>
</tr>
<tr>
<td>2010-41</td>
<td>Guillaume Chaslot (UM)</td>
<td>Monte-Carlo Tree Search</td>
</tr>
<tr>
<td>2010-42</td>
<td>Sybren de Kinderen (VU)</td>
<td>Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach</td>
</tr>
<tr>
<td>2010-43</td>
<td>Peter van Kranenburg (UU)</td>
<td>A Computational Approach to Content-Based Retrieval of Folk Song Melodies</td>
</tr>
<tr>
<td>2010-44</td>
<td>Pieter Bellekens (TUE)</td>
<td>An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain</td>
</tr>
<tr>
<td>2010-45</td>
<td>Vasilios Andrikopoulos (UvT)</td>
<td>A theory and model for the evolution of software services</td>
</tr>
<tr>
<td>2010-46</td>
<td>Vincent Pijpers (VU)</td>
<td>e3alignment: Exploring Inter-Organizational Business-ICT Alignment</td>
</tr>
<tr>
<td>2010-47</td>
<td>Chen Li (UT)</td>
<td>Mining Process Model Variants: Challenges, Techniques, Examples</td>
</tr>
<tr>
<td>2010-48</td>
<td>Milan Lovric (EUR)</td>
<td>Behavioral Finance and Agent-Based Artificial Markets</td>
</tr>
<tr>
<td>2010-49</td>
<td>Jahn-Takeshi Saito (UM)</td>
<td>Solving difficult game positions</td>
</tr>
<tr>
<td>2010-50</td>
<td>Bouke Huurnink (UvA)</td>
<td>Search in Audiovisual Broadcast Archives</td>
</tr>
<tr>
<td>2010-51</td>
<td>Alia Khairia Amin (CWI)</td>
<td>Understanding and supporting information seeking tasks in multiple sources</td>
</tr>
<tr>
<td>2010-52</td>
<td>Peter-Paul van Maanen (VU)</td>
<td>Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention</td>
</tr>
<tr>
<td>2011-01</td>
<td>Botond Cseke (RUN)</td>
<td>Combining Concepts and Language Models for Information Access</td>
</tr>
<tr>
<td>2011-02</td>
<td>Nick Timmermeier (UU)</td>
<td>Variational Algorithms for Bayesian Inference in Latent Gaussian Models</td>
</tr>
<tr>
<td>2011-03</td>
<td>Jan Martijn van der Werf (TUE)</td>
<td>Work flows in Life Science</td>
</tr>
<tr>
<td>2011-04</td>
<td>Jan Martijn van der Werf (TUE)</td>
<td>Compositional Design and Verification of Component-Based Information Systems</td>
</tr>
<tr>
<td>2011-05</td>
<td>Hado van Hasselt (UU)</td>
<td>Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms</td>
</tr>
<tr>
<td>2011-06</td>
<td>Bas van der Raadt (VU)</td>
<td>Enterprise Architecture Coming of Age – Increasing the Performance of an Emerging Discipline</td>
</tr>
<tr>
<td>2011-07</td>
<td>Niels Lohmann (TUE)</td>
<td>Semantically-Enhanced Recommendations in Cultural Heritage</td>
</tr>
<tr>
<td>2011-08</td>
<td>Yujia Cao (UT)</td>
<td>Multimodal Information Presentation for High Load Human Computer Interaction</td>
</tr>
<tr>
<td>2011-09</td>
<td>Tim de Jong (UU)</td>
<td>BDI-based Generation of Robust Task-Oriented Dialogues</td>
</tr>
<tr>
<td>2011-10</td>
<td>Bart Bogaut (UvT)</td>
<td>Contextualised Mobile Media for Learning</td>
</tr>
<tr>
<td>2011-11</td>
<td>Dhaval Vyas (UT)</td>
<td>Cloud Content Contention</td>
</tr>
<tr>
<td>2011-12</td>
<td>Carmen Bratosin (TUE)</td>
<td>Designing for Awareness: An Experience-focused HCI Perspective</td>
</tr>
<tr>
<td>2011-13</td>
<td>Xiaoyu Mao (UvT)</td>
<td>Grid Architecture for Distributed Process Mining</td>
</tr>
<tr>
<td>2011-14</td>
<td>Xiangyu Hao (UT)</td>
<td>Airport under Control. Multiagent Scheduling for Airport Ground Handling</td>
</tr>
</tbody>
</table>
2011-14 Milan Lovric (EUR)  
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)  
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)  
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UvA)  
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)  
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)  
The Mind’s Eye on Personal Profiles

2011-20 Qing Gu (VU)  
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)  
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UvA)  
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UvA)  
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)  
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Quonain Jaffry (VU)  
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)  
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)  
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein (UvA)  
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)  
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)  
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)  
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)  
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)  
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)  
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)  
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)  
Experiments in serious game design: a cognitive approach

2011-37 Adriana Birlutiu (RUN)  
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)  
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)  
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)  
Architectural Knowledge Management in Global Software Development


Bibliography


Bibliography


Bibliography


