

# COORDINATING REQUIREMENTS ENGINEERING AND SOFTWARE TESTING

MICHAEL UNTERKALMSTEINER



Department of Software Engineering doctoral dissertation series No  
2015:08

# COORDINATING REQUIREMENTS ENGINEERING AND SOFTWARE TESTING

MICHAEL UNTERKALMSTEINER

Doctoral Dissertation in Software Engineering



Department of Software Engineering  
Blekinge Institute of Technology  
SWEDEN

2015 Michael Unterkalmsteiner  
Department of Software Engineering  
Publisher: Blekinge Institute of Technology,  
SE-371 79 Karlskrona, Sweden  
Printed by Lenanders Grafiska AB, 2015  
ISBN: 978-91-7295-306-2  
ISSN: 1653-2090

To my parents.

There is not a discovery in  
science, however revolutionary,  
however sparkling with insight,  
that does not arise out of what  
went before.

---

*Adding a Dimension* (1966)  
ISAAC ASIMOV



---

## ABSTRACT

---

The development of large, software-intensive systems is a complex undertaking that is generally tackled by a divide and conquer strategy. Organizations face thereby the challenge of coordinating the resources which enable the individual aspects of software development, commonly solved by adopting a particular process model. The alignment between requirements engineering (RE) and software testing (ST) activities is of particular interest as those two aspects are intrinsically connected: requirements are an expression of user/customer needs while testing increases the likelihood that those needs are actually satisfied.

The work in this thesis is driven by empirical problem identification, analysis and solution development towards two main objectives. The first is to develop an understanding of RE and ST alignment challenges and characteristics. Building this foundation is a necessary step that facilitates the second objective, the development of solutions relevant and scalable to industry practice that improve REST alignment.

The research methods employed to work towards these objectives are primarily empirical. Case study research is used to elicit data from practitioners while technical action research and field experiments are conducted to validate the developed solutions in practice.

This thesis contains four main contributions: (1) An in-depth study on REST alignment challenges and practices encountered in industry. (2) A conceptual framework in the form of a taxonomy providing constructs that further our understanding of REST alignment. The taxonomy is operationalized in an assessment framework, REST-bench (3), that was designed to be lightweight and can be applied as a postmortem in closing development projects. (4) An extensive investigation into the potential of information retrieval techniques to improve test coverage, a common REST alignment challenge, resulting in a solution prototype, risk-based testing supported by topic models (RiTTM).

REST-bench has been validated in five cases and has shown to be efficient and effective in identifying improvement opportunities in the coordination of RE and ST. Most of the concepts operationalized from the REST taxonomy were found to be useful, validating the conceptual framework. RiTTM, on the other hand, was validated in a single case experiment where it has shown great potential, in particular by identifying test cases that were originally overlooked by expert test engineers, improving effectively test coverage.



---

## ACKNOWLEDGMENTS

---

I have started and finished this thesis alone, however met a bunch of people along the way (or left them). This section is dedicated to them.

There is little in this work that has not been directly or indirectly influenced by my two supervisors, Tony Gorschek and Robert Feldt. Tony has found the right words, at the right time, when I needed an impulse (in any direction). This makes him a great motivator and mentor. Robert's feedback, be it in written communication or in lively discussions, has always been to the point, insightful and constructive. This makes him a great peer and inspirer.

SERL Sweden provided a supportive work environment, forming a group that is diverse and at the same time focused on high quality research. I would like to thank Mikael for the work related discussions, but even more so for the nerd-talk that lightened up my day. Nauman and Bogdan have always been excellent colleagues that, besides being supportive, made my stay at the office a little less about work and a little more about life. This is actually true for all colleagues at SERL Sweden. Furthermore, I would like to thank all my co-authors.

This research would not have been possible without the support from the companies that provided their time and effort. I would like to thank Louise, Antonios and Johan for their collaboration and patience in trying out my ideas. My appreciation goes to the engineers whose brains I could pick.

I value nothing higher than the support I received from my parents, my sister, brother and my grandmas, not only during the last five years but in all the ups and downs before, being physically close or far away from them. You're all in my thoughts, very often.

My wife, Rocio, is the haven I return to after a day at the job. Nothing I have accomplished would have been possible without her support and tolerance for my endeavor.

Karlskrona, May 1, 2015



---

## PREFACE

---

### PAPERS IN THIS THESIS

This compilation thesis includes the following six papers<sup>1</sup>.

**Chapter 2:** Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt, "Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1809–1855, 2014.

**Chapter 3:** Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek, "A Taxonomy for Requirements Engineering and Software Test Alignment," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, 2014.

**Chapter 4:** Michael Unterkalmsteiner, Tony Gorschek, Robert Feldt, and Eriks Klotins, "Assessing Requirements Engineering and Software Test Alignment with REST-bench – Five Case Studies," *Journal of Systems and Software*, Under Review, 2015.

**Chapter 5:** Elizabeth Bjarnason, Michael Unterkalmsteiner, Emelie Engström, and Markus Borg, "An Industrial Case Study on the Use of Test Cases as Requirements," in *Proceedings of the 16th International Conference on Agile Software Development*, Helsinki, Finland, 2015.

**Chapter 6:** Michael Unterkalmsteiner, Tony Gorschek, Robert Feldt, and Niklas Lavesson, "Large-scale Information Retrieval in Software Engineering – An Experience Report from Industrial Application," *Empirical Software Engineering*, Under Review, 2015.

**Chapter 7:** Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek, "Supporting Experts in Test Case Selection with Topic Models," *Software: Testing, Verification and Reliability*, Under Submission, 2015.

### CONTRIBUTION STATEMENT

Michael Unterkalmsteiner is the lead author of all but Chapter 2 and Chapter 5 in this thesis. As lead author, he took the main responsibility in de-

---

<sup>1</sup> Papers marked with "Under Review" are currently considered for publication. Papers marked with "Under Submission" have been submitted but were not reviewed yet at the time of this writing.

signing the studies, collecting and analyzing data, and reporting the findings in peer-reviewed publications. Furthermore, he is the sole author of Chapter 1, the introduction. His contribution to Chapter 2 and Chapter 5, and the co-authors' contribution in the remaining chapters are described next.

**CHAPTER 2:** Michael Unterkalmsteiner participated in the data collection, performing three interviews, transcribing seven and coding eleven (of 30). He participated in six workshops where the data analysis with the other authors were discussed. He reviewed and commented on the final draft of the paper.

**CHAPTER 3:** Robert Feldt and Tony Gorschek contributed the idea to define a taxonomy of REST alignment methods. Both participated in the development by reviewing intermediate states of the taxonomy and providing comments on the taxonomy's structure. Tony Gorschek had the idea to operationalize the taxonomy as an assessment framework and provided the contact to the company where the pilot evaluation of REST-bench was performed. Both reviewed and commented on intermediate versions and the final draft of the paper.

**CHAPTER 4:** Tony Gorschek and Robert Feldt provided the company contacts where REST-bench was evaluated. Both reviewed and commented on the final draft of the paper. Eriks Klotins developed the web-based artifact mapping prototype.

**CHAPTER 5:** The data in this paper originates from the same case study reported in Chapter 2. In addition to the original data collection, Michael Unterkalmsteiner wrote large parts of the related work and contributed to discussion sections where the research questions are answered. He reviewed and commented intermediate versions and the final draft of the paper.

**CHAPTER 6:** The idea of the study was discussed with all co-authors. Details of the IR technique evaluation were discussed both with Niklas Lavesson and Robert Feldt. Tony Gorschek, Robert Feldt and Niklas Lavesson reviewed and commented on an intermediate version and the final draft of the paper.

**CHAPTER 7:** The idea of the study was discussed with both Robert Feldt and Tony Gorschek. Both reviewed and commented on the final draft of the paper.

## RELATED PAPERS NOT INCLUDED IN THIS THESIS

- Giedre Sabaliauskaite, Annabella Loconsole, Emelie Engström, Michael Unterkalmsteiner, Björn Regnell, Per Runeson, Tony Gorschek, and Robert Feldt, "Challenges in aligning requirements engineering and verification in a large-scale industrial context," in Proceedings 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany, 2010, pp. 128–142.
- Michael Unterkalmsteiner, Tony Gorschek, A. K. M. Moinul Islam, Chow K. Cheng, Rahadian B. Permadi, and Robert Feldt, "Evaluation and Measurement of Software Process Improvement – A Systematic Literature Review," IEEE Transactions on Software Engineering, vol. 38, no. 2, pp. 398–424, Apr. 2012.
- Michael Unterkalmsteiner, Tony Gorschek, A. K. M. Moinul Islam, Chow K. Cheng, Rahadian B. Permadi, and Robert Feldt, "A conceptual framework for SPI evaluation," Journal of Software: Evolution and Process, vol. 26, no. 2, pp. 251–279, 2014.

## OTHER PAPERS NOT INCLUDED IN THIS THESIS

- Sebastian Barney, Mahvish Khurum, Kai Petersen, Michael Unterkalmsteiner, and Ronald Jabangwe, "Improving Students With Rubric-Based Self-Assessment and Oral Feedback," IEEE Transactions on Education, vol. 55, no. 3, pp. 319–325, Aug. 2012.
- Nicoló Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson, "Software Development in Startup Companies: A Systematic Mapping Study," Information and Software Technology, vol. 56, no. 10, pp. 1200–1218, 2014.
- Carmine Giardino, Michael Unterkalmsteiner, Nicoló Paternoster, Tony Gorschek, and Pekka Abrahamsson, "What do we know about software development in startups?," IEEE Software, vol. 31, no. 5, pp. 28–32, 2014.
- Nauman bin Ali and Michael Unterkalmsteiner, "Use and evaluation of simulation for software process education: a case study," in European Conference Software Engineering Education (ECSEE), Seeon Monastery, Germany, 2014.
- Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek, "Software Engineering Practices in Start-up Companies: A Mapping Study," in 6th International Conference on Software Business, 2015.
- Carmine Giardino, Nicoló Paternoster, Michael Unterkalmsteiner, and Tony Gorschek, "Software Development in Startup Companies:

The Greenfield Startup Model,” IEEE Transactions on Software Engineering, Under Review, 2015.

#### FUNDING

The research in this thesis was funded by EASE Industrial Excellence Center for Embedded Applications Software Engineering (<http://ease.cs.lth.se>).

---

## CONTENTS

---

|  |           |
|--|-----------|
| Abstract   | vii       |
| Acknowledgments  | ix        |
| Preface  | xi        |
| <b>1 INTRODUCTION</b>  | <b>1</b>  |
| 1 Overview . . . . .   | 1         |
| 2 Background and Related Work . . . . .  | 4         |
| 2.1 Coordination in Software Development . . . . .   | 4         |
| 2.2 Test Case Selection . . . . .  | 5         |
| 2.3 Traceability Recovery . . . . .  | 6         |
| 3 Research Questions . . . . .   | 6         |
| 3.1 RG-I: Understand REST Alignment . . . . .  | 8         |
| 3.2 RG-II: Improve REST Alignment . . . . .  | 9         |
| 4 Research Approach . . . . .  | 10        |
| 4.1 Research Methods in General . . . . .  | 11        |
| 4.2 Research Framework in this Thesis . . . . .  | 15        |
| 4.3 Research Setting and Data Collection . . . . .   | 19        |
| 4.4 Limitations . . . . .  | 21        |
| 5 Overview of the Chapters . . . . .   | 22        |
| 5.1 Chapter 2 . . . . .  | 22        |
| 5.2 Chapter 3 . . . . .  | 23        |
| 5.3 Chapter 4 . . . . .  | 26        |
| 5.4 Chapter 5 . . . . .  | 27        |
| 5.5 Chapter 6 . . . . .  | 28        |
| 5.6 Chapter 7 . . . . .  | 29        |
| 6 Synthesis . . . . .  | 29        |
| 6.1 RG-I: Understand REST Alignment . . . . .  | 30        |
| 6.2 RG-II: Improve REST Alignment . . . . .  | 33        |
| 7 Conclusions and Future Work . . . . .  | 35        |
| <b>2 CHALLENGES AND PRACTICES IN ALIGNING REQUIREMENTS WITH VERIFICATION AND VALIDATION: A CASE STUDY OF SIX COMPANIES</b> | <b>37</b> |
| 1 Introduction . . . . .   | 37        |
| 2 Related Work . . . . .   | 39        |
| 3 Case Study Design . . . . .  | 42        |
| 3.1 Definition of Research Goal and Questions . . . . .  | 44        |
| 3.2 Design and Planning . . . . .  | 44        |
| 3.3 Evidence Collection . . . . .  | 47        |
| 3.4 Data Analysis . . . . .  | 48        |

|     |   |     |
|-----|---|-----|
| 3.5 | Threats to Validity . . . . .   | 51  |
| 4   | Results . . . . .   | 54  |
| 4.1 | Alignment Challenges . . . . .  | 55  |
| 4.2 | Practices for Improved Alignment . . . . .  | 67  |
| 4.3 | Practices that Address the Challenges . . . . .   | 78  |
| 5   | Discussion . . . . .  | 82  |
| 6   | Conclusions . . . . .   | 87  |
| 3   | <b>A TAXONOMY FOR REQUIREMENTS ENGINEERING AND SOFTWARE TEST ALIGNMENT</b>                                | 89  |
| 1   | Introduction . . . . .  | 89  |
| 2   | Background and Related Work . . . . .   | 91  |
| 2.1 | The Need for Alignment . . . . .  | 91  |
| 2.2 | Alignment vs. Traceability . . . . .  | 93  |
| 2.3 | The Purpose of Taxonomies . . . . .   | 93  |
| 2.4 | Taxonomies in Software Engineering . . . . .  | 94  |
| 2.5 | Developing Taxonomies . . . . .   | 94  |
| 3   | The REST Taxonomy . . . . .   | 95  |
| 3.1 | Relevance . . . . .   | 97  |
| 3.2 | The Information Dyad . . . . .  | 98  |
| 3.3 | Method Context . . . . .  | 104 |
| 3.4 | Dyad Structure Properties . . . . .   | 105 |
| 3.5 | Method Classification . . . . .   | 108 |
| 4   | Taxonomy Construction and Validation Method . . . . .   | 111 |
| 4.1 | Iterative construction and validation . . . . .   | 111 |
| 4.2 | Validity Threats . . . . .  | 115 |
| 5   | Method Evaluation using the REST Taxonomy . . . . .   | 116 |
| 5.1 | Summary Analysis . . . . .  | 116 |
| 5.2 | Dyad Structure Analysis . . . . .   | 117 |
| 5.3 | Lessons Learned and Limitations . . . . .   | 120 |
| 6   | Industrial Case Study using REST-bench . . . . .  | 122 |
| 6.1 | REST-bench Process Overview . . . . .   | 122 |
| 6.2 | Case Study: REST-bench Results . . . . .  | 125 |
| 6.3 | Identifying improvements using REST-bench . . . . .   | 128 |
| 6.4 | Lessons Learned using REST-bench in Industry and Limitations . . . . .                                    | 129 |
| 7   | Conclusions and Future Work . . . . .   | 130 |
| 4   | <b>ASSESSING REQUIREMENTS ENGINEERING AND SOFTWARE TEST ALIGNMENT WITH REST-BENCH – FIVE CASE STUDIES</b> | 137 |
| 1   | Introduction . . . . .  | 137 |
| 2   | Background and Related Work . . . . .   | 139 |
| 2.1 | REST Alignment . . . . .  | 139 |
| 2.2 | The REST Taxonomy . . . . .   | 139 |
| 2.3 | Related Work . . . . .  | 140 |

|     |  |     |
|-----|--|-----|
| 3   | Research Method . . . . .  | 142 |
| 3.1 | Project Characteristics . . . . .  | 142 |
| 3.2 | Data Collection . . . . .  | 144 |
| 3.3 | Limitations . . . . .  | 144 |
| 4   | REST-bench . . . . .   | 145 |
| 4.1 | Step 1 – Selection . . . . .   | 146 |
| 4.2 | Step 2 – Data Elicitation . . . . .  | 147 |
| 4.3 | Step 3 – Map Construction . . . . .  | 149 |
| 4.4 | Step 4 – Assessment Workshop . . . . .   | 153 |
| 4.5 | Step 5 – Report Recommendations . . . . .  | 157 |
| 5   | Case Studies . . . . .   | 158 |
| 5.1 | Company A . . . . .  | 158 |
| 5.2 | Company B . . . . .  | 161 |
| 5.3 | Company C . . . . .  | 163 |
| 5.4 | Company D . . . . .  | 165 |
| 6   | Discussion . . . . .   | 168 |
| 6.1 | RQ1: To what extent are the dyad structures from the REST taxonomy useful to elicit improvement opportunities? . . . . . | 168 |
| 6.2 | RQ2: To what extent is REST-bench in Agile and plan-driven environments useful? . . . . .                                | 171 |
| 6.3 | RQ3: Is REST-bench usable? . . . . .   | 172 |
| 7   | Conclusions . . . . .  | 173 |
| 5   | <b>AN INDUSTRIAL CASE STUDY ON USING TEST CASES AS REQUIREMENTS</b>  | 175 |
| 1   | Introduction . . . . .   | 175 |
| 2   | Agile RE: Test Cases as Requirements Documentation . . . . .   | 176 |
| 3   | Case Companies . . . . .   | 177 |
| 4   | Method . . . . .   | 179 |
| 5   | Results . . . . .  | 179 |
| 5.1 | Company A: A De Facto Practice . . . . .   | 180 |
| 5.2 | Company B: An Established Practice . . . . .   | 180 |
| 5.3 | Company F: Planned Practice as part of Agile Transition . . . . .  | 182 |
| 5.4 | Limitations . . . . .  | 182 |
| 6   | Test Cases in the Role of Requirements (RQ1) . . . . .   | 183 |
| 6.1 | Elicitation and Validation . . . . .   | 183 |
| 6.2 | Verification . . . . .   | 184 |
| 6.3 | Tracing . . . . .  | 185 |
| 6.4 | Requirements Management . . . . .  | 185 |
| 7   | The Reasons for and Contexts of the Practice (RQ2) . . . . .   | 186 |
| 8   | Conclusions and Future Work . . . . .  | 186 |

|          |   |            |
|----------|---|------------|
| <b>6</b> | <b>LARGE-SCALE INFORMATION RETRIEVAL IN SOFTWARE ENGINEERING – AN EXPERIENCE REPORT FROM INDUSTRIAL APPLICATION</b> | <b>189</b> |
| 1        | Introduction . . . . .  | 189        |
| 2        | Background and Related Work . . . . .   | 191        |
| 2.1      | Case Context . . . . .  | 191        |
| 2.2      | Overview of the State of Art . . . . .  | 193        |
| 2.3      | Solution Development . . . . .  | 194        |
| 2.4      | Related Work . . . . .  | 196        |
| 3        | Research Method . . . . .   | 198        |
| 3.1      | Description of Objects . . . . .  | 199        |
| 3.2      | Feature Chains . . . . .  | 199        |
| 3.3      | Independent Variables . . . . .   | 200        |
| 3.4      | Validity Threats . . . . .  | 202        |
| 4        | Experimental Setups and Evaluation . . . . .  | 203        |
| 4.1      | Experimental Setup 1: Pilot Experiment . . . . .  | 204        |
| 4.2      | Experimental Setup 2: Scaling Up . . . . .  | 212        |
| 4.3      | Experimental Setup 3: Ranking . . . . .   | 219        |
| 5        | Conclusions and Future Work . . . . .   | 230        |
| <b>7</b> | <b>SUPPORTING EXPERTS IN TEST CASE SELECTION WITH TOPIC MODELS</b>  | <b>233</b> |
| 1        | Introduction . . . . .  | 233        |
| 2        | Background and Related Work . . . . .   | 235        |
| 2.1      | Risk-based Testing . . . . .  | 235        |
| 2.2      | Test Case Selection . . . . .   | 236        |
| 2.3      | Topic Models . . . . .  | 237        |
| 2.4      | Topic Models in Software Engineering . . . . .  | 240        |
| 2.5      | Contribution . . . . .  | 240        |
| 3        | Risk-based testing supported by topic models (RiTMM) . . . . .  | 241        |
| 3.1      | Step 1 – Model Selection . . . . .  | 243        |
| 3.2      | Step 2 – Topic Filter Creation . . . . .  | 244        |
| 3.3      | Step 3 – Test Case Selection Decision Support . . . . .   | 246        |
| 4        | Case Study Design . . . . .   | 246        |
| 4.1      | Case Context . . . . .  | 246        |
| 4.2      | Implementation of RiTTM . . . . .   | 248        |
| 4.3      | Evaluation of RiTTM . . . . .   | 252        |
| 4.4      | Validity Threats . . . . .  | 253        |
| 5        | Case Study Results . . . . .  | 254        |
| 5.1      | Model Selection Results . . . . .   | 254        |
| 5.2      | Test Case Selection Results . . . . .   | 256        |
| 6        | Conclusions . . . . .   | 259        |
| 6.1      | Future Work . . . . .   | 259        |
|          | <b>BIBLIOGRAPHY</b>   | <b>261</b> |

---

## LIST OF FIGURES

---

|            |  |     |
|------------|--|-----|
| Figure 1.1 | Thesis focus . . . . .                                 | 2   |
| Figure 1.2 | Research questions answered in this thesis . . . . .   | 7   |
| Figure 1.3 | Research worldviews . . . . .                          | 10  |
| Figure 1.4 | Design science framework . . . . .                     | 16  |
| Figure 1.5 | Thesis research framework . . . . .                    | 17  |
| Figure 1.6 | Information dyad and dyad structure . . . . .          | 24  |
| Figure 1.7 | Dyad structure evolution at Company A . . . . .        | 31  |
| Figure 2.1 | Overview of the research process . . . . .             | 43  |
| Figure 2.2 | Conceptual model of the area under study . . . . .     | 45  |
| Figure 2.3 | Example of the interviewee data abstraction . . . . .  | 50  |
| Figure 2.4 | Variation factors of studied companies . . . . .       | 85  |
| Figure 3.1 | Anatomy of the REST taxonomy . . . . .                 | 95  |
| Figure 3.2 | Cases classified in the REST taxonomy . . . . .        | 110 |
| Figure 3.3 | Taxonomy construction and validation process . . . . . | 112 |
| Figure 3.4 | REST-bench artifact maps from the case study . . . . . | 126 |
| Figure 4.1 | Information dyad and dyad structure . . . . .          | 139 |
| Figure 4.2 | REST-bench steps . . . . .                             | 145 |
| Figure 4.3 | Artifact elicitation example . . . . .                 | 150 |
| Figure 4.4 | Artifact map – Illustration of components . . . . .    | 151 |
| Figure 4.5 | Artifact map of Case E . . . . .                       | 156 |
| Figure 4.6 | Artifact map of Case A . . . . .                       | 159 |
| Figure 4.7 | Artifact map of Case B . . . . .                       | 162 |
| Figure 4.8 | Artifact map of Case C . . . . .                       | 163 |
| Figure 4.9 | Artifact map of Case D . . . . .                       | 167 |
| Figure 6.1 | Example chain . . . . .                                | 200 |
| Figure 6.2 | Experimental setups . . . . .                          | 204 |
| Figure 6.3 | Pilot setup details . . . . .                          | 205 |
| Figure 6.4 | Pilot experiment results . . . . .                     | 208 |
| Figure 6.5 | Results on three complete product variants . . . . .   | 215 |
| Figure 6.6 | SVD runtime and dimensions . . . . .                   | 217 |
| Figure 6.7 | Ranking setup . . . . .                                | 218 |
| Figure 6.8 | Box plots text case rankings . . . . .                 | 224 |
| Figure 7.1 | Topic model example . . . . .                          | 238 |
| Figure 7.2 | The RiTTM method . . . . .                             | 241 |
| Figure 7.3 | Topic filter table implementation . . . . .            | 245 |
| Figure 7.4 | Testing process for new product development . . . . .  | 248 |
| Figure 7.5 | Topic proportion selection . . . . .                   | 251 |
| Figure 7.6 | Model precision . . . . .                              | 255 |



---

## LIST OF TABLES

---

|            |  |     |
|------------|--|-----|
| Table 1.1  | Empirical research method characteristics . . . . .      | 12  |
| Table 1.2  | Companies contributing to the thesis . . . . .           | 20  |
| Table 1.3  | Dyad structure properties . . . . .                      | 25  |
| Table 2.1  | Characteristics of the case study companies . . . . .    | 46  |
| Table 2.2  | Overview of interviewees' roles . . . . .                | 48  |
| Table 2.3  | Alignment challenges . . . . .                           | 56  |
| Table 2.4  | Alignment practices . . . . .                            | 68  |
| Table 2.5  | Tool usage . . . . .                                     | 76  |
| Table 2.6  | Alignment practices mapped to challenges . . . . .       | 79  |
| Table 3.1  | REST classification process . . . . .                    | 97  |
| Table 3.2  | Example trade-off analysis . . . . .                     | 118 |
| Table 3.3  | Context of Case A . . . . .                              | 133 |
| Table 3.4  | Dyad structure and characterization of Case A . . . . .  | 133 |
| Table 3.5  | Context of Case B . . . . .                              | 134 |
| Table 3.6  | Dyad structure and characterization of Case B . . . . .  | 134 |
| Table 3.7  | Context of Case C . . . . .                              | 135 |
| Table 3.8  | Dyad structure and characterization of Case C . . . . .  | 135 |
| Table 3.9  | Context of Case D . . . . .                              | 136 |
| Table 3.10 | Dyad structure and characterization of Case D . . . . .  | 136 |
| Table 4.1  | Dyad structure properties . . . . .                      | 140 |
| Table 4.2  | Project characteristics . . . . .                        | 143 |
| Table 4.3  | Example artifacts . . . . .                              | 148 |
| Table 4.4  | Seeding questions . . . . .                              | 152 |
| Table 4.5  | Post assessment questionnaire . . . . .                  | 169 |
| Table 4.6  | Usability assessment of REST-bench . . . . .             | 172 |
| Table 5.1  | Overview of interviewees . . . . .                       | 178 |
| Table 5.2  | Summary of benefits and challenges . . . . .             | 184 |
| Table 6.1  | Candidate artifacts . . . . .                            | 194 |
| Table 6.2  | Selected independent variables (IV) and values . . . . . | 201 |
| Table 6.3  | VSM results using the Dsim measure . . . . .             | 214 |
| Table 6.4  | Large scale IR experiments on source code . . . . .      | 216 |
| Table 6.5  | Corpora sizes (# of documents / # of terms) . . . . .    | 222 |
| Table 6.6  | Randomization test results . . . . .                     | 224 |
| Table 6.7  | Test case ranking positions . . . . .                    | 225 |
| Table 6.8  | Quality of the ranking results . . . . .                 | 226 |
| Table 7.1  | Test case selection results . . . . .                    | 257 |



# 1

---

## INTRODUCTION

---

### 1 OVERVIEW

Software requirements identify the needs and limitations placed on a software product that aims to solve a real-world problem [Kotonya and Sommerville, 1998]. Requirements engineering denotes the systematic handling of those requirements including their elicitation, analysis, prioritization, validation and maintenance [Bourque and Fairley, 2014]. Software testing, on the other hand, denotes a process determining whether the developed product fulfills its intended purpose and meets the identified software requirements [Bourque and Fairley, 2014].

The requirements, defining a real-world problem or need, and the verification of a product that provides a solution to that problem or need, are two sides of the same coin which require alignment. With alignment we mean *the adjustment of requirements engineering (RE) and software testing (ST) activities for coordinated functioning and optimized product development<sup>1</sup>* [Unterkalmsteiner et al., 2014a]. We argue that alignment between RE and ST activities is of paramount importance for developing software intensive systems [ISO/IEC, 2007], since:

- Requirements tend to change over time [Stark et al., 1999, Lam and Shankararaman, 1999, Nurmuliani et al., 2004], for various reasons [McGee and Greer, 2012], potentially leading to inconsistencies in test plans and test cases, and reducing test coverage of the requirements.
- Requirements specifications may be incomplete or ambiguous [Bach, 1999, Damian and Chisan, 2006], leading to implicit assumptions during development [Albayrak et al., 2009] and limiting the requirements-based verification of the developed product.
- Each unit increase of software requirements complexity [Regnell et al., 2008] quadruples the complexity of the developed solution [Glass, 2002b], increasing also the verification cost [Tassey, 2002].

---

<sup>1</sup> Due to this definition, we use the terms “alignment” and “coordination” interchangeably in the remainder of this thesis.

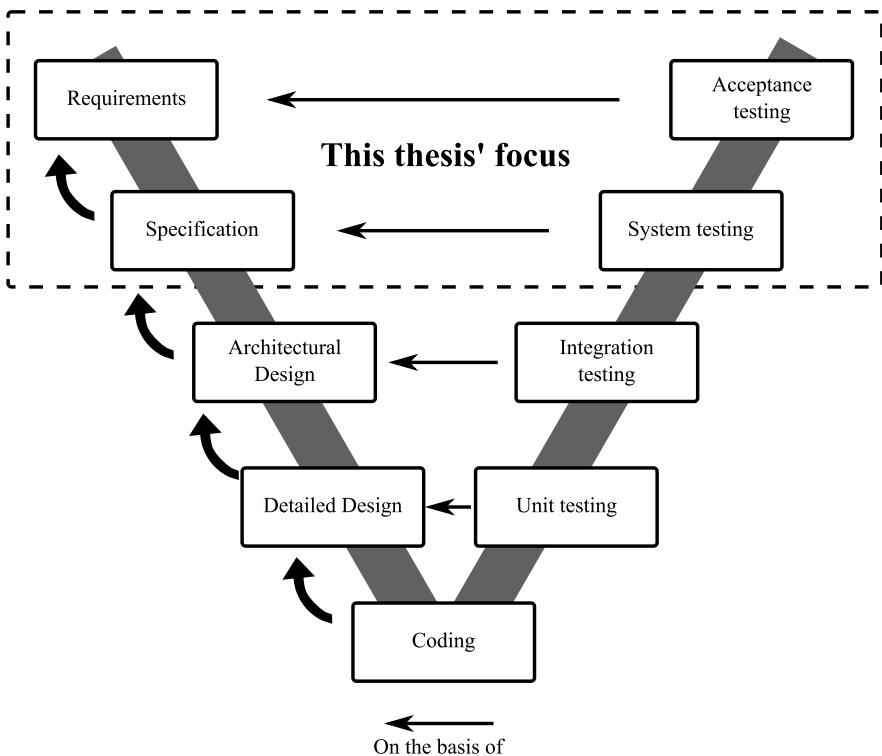


Figure 1.1: Thesis focus illustrated in the V-Model

From a process perspective, software development consists of transitions from system concept, requirements specification, analysis and design, implementation, and test and maintenance [Laplante, 2007, p. 23]. This is an abstraction that applies to both plan driven process models (e.g. spiral [Boehm, 1988] and evolutionary [Naumann and Jenkins, 1982], and the unified process model [Kruchten, 2000]), as well as and Agile models, although to a lesser extent in the latter category as activities may be blended, eliminating transitions altogether (e.g. in eXtreme Programming [Beck, 1999]).

Figure 1.1 illustrates the V-Model of software development<sup>2</sup>, which originates from system engineering [Forsberg and Mooz, 1991, Bröhl and Dröscher, 1995] and was adopted in software engineering [Pfleeger and Atlee, 2009]. This model illustrates vertical transitions between software development activities (left hand side) and the corresponding testing activities (right hand side) that ought to ensure the quality of the resulting

<sup>2</sup> Note that this is an oversimplification, but it serves the purpose of introducing the context of this thesis

work products. The importance of enabling these vertical transitions and align the intentions and activities across is demonstrated by an abundance of research (e.g. between requirements abstraction levels [Gorschek and Wohlin, 2006], requirements and software architecture/design [Kop and Mayr, 1998, Amyot and Mussbacher, 2001, Hall et al., 2002], software architecture/design and implementation [Murphy et al., 2001, Elrad et al., 2002, Aldrich et al., 2002], and software architecture/design and testing [Muccini et al., 2004, Samuel et al., 2007]).

However, aligning requirements engineering and software testing is a less explored territory, even though it would be beneficial to recognize the inherent link between them [Graham, 2002]. Therefore, it is essential that we increase our understanding through the study of these connections, and treat them as a collective and not as individual, isolated areas of research.

We began our exploration of REST alignment by studying the phenomenon in its real-life setting through a large case study [Yin, 2003]. This investigation enabled us to elicit concrete challenges encountered in industry, related to the coordination of requirements engineering and software testing (Chapter 2). While these results were useful to define the scope of the research, we were still lacking the tools to characterize REST alignment at a level that allows at least a rudimentary analysis of alignment, or lack thereof. We addressed this gap in Chapter 3 by developing a taxonomy, which has been a scientific tool since the early work by Carl von Linné [Linnaei, 1735]. The main result of this classification exercise was the definition of the information dyad, a construct that we used to characterize alignment methods but also to assess the state of REST alignment. Based on this work, we developed REST-bench as a project postmortem [Birk et al., 2002] assessment process to efficiently identify misalignment and pointing out concrete improvement possibilities (Chapter 4).

In a parallel track of work, we had observed how REST alignment was seemingly addressed in some companies by reusing test cases as requirements (Chapter 5). While this practice has its benefits, it leads also to challenges in test case selection [Rothermel and Harrold, 1996], that we aimed to address in the remainder of this thesis. In Chapter 6, we evaluated varying configurations of information retrieval techniques [Grossman and Frieder, 2004], aiming at recovering traces between source code and test cases that would allow test engineers to select test cases based on particular product configurations. While the approach worked in principle, we encountered an array of practical challenges in implementing it on industry-grade data sets. Therefore, we re-designed the approach in Chapter 7, focusing however on supporting engineers to improve test coverage of those product features which were exposed to a high failure risk [Felderer and Schieferdecker, 2014]. In summary, this thesis contributes to the knowledge on coordinating requirements engineering (RE) and software testing (ST) by:

- An in-depth study on REST alignment challenges and practices encountered in industry.
- A conceptual framework, in the form of a taxonomy, that provides a mean to characterize REST alignment methods.
- A lightweight assessment framework, REST-bench, aimed at identifying REST alignment improvement opportunities.
- An extensive investigation into the potential of information retrieval techniques to improve test coverage, a common REST alignment challenge, resulting in a solution prototype, risk-based testing supported by topic models (RiTMM).

The remainder of this chapter is structured as follows. Section 2 introduces the necessary background to follow the line of thought in this chapter. We introduce our research goals and discuss the driving questions in Section 3 while the research approach to answer them is presented in Section 4. The results and main findings of the studies contained in this thesis are summarized in Section 5. In Section 6 we synthesize the individual results into a discussion addressing our overall research goals. We conclude this chapter in Section 7, pointing out avenues for future work.

## 2 BACKGROUND AND RELATED WORK

We introduce here the concepts required to follow the argumentation in the remainder of this chapter. A more detailed account on background and related work is given in the respective chapters of this thesis.

### 2.1 *Coordination in Software Development*

Kraut and Streeter [1995] identify two factors that render the coordination in software development challenging. The scale of a software development project determines to what extent labor is divided, leading to workforce specialization and compartmentalization, and consequentially to an increased need for coordination [Kraut and Streeter, 1995]. Coordination is also affected by the uncertainty of a software development project, stemming from the varying performance of engineers, changing and incomplete requirements, and varying interpretations of customer needs [Kraut and Streeter, 1995]. Practices such as cross-functional teams [Marczak and Damian, 2011] or job rotation [Faegri et al., 2010] are organizational means to address compartmentalization. Research on agile development practices [Beck et al., 2001] also suggests that they enhance coordination [Begel and Nagappan, 2007, Mishra et al., 2012] by embracing uncertainty and managing change [Dybå and Dingsoyr, 2008]; however, scaling Agile practices to large organizations leads also to management overhead of the agile teams,

requiring much coordination and communication [Petersen and Wohlin, 2010].

Herbsleb and Grinter [1999] also describe the challenges of coordinating development teams, operating however at different sites. They illustrate the limitations of explicit coordination mechanisms such as plans, interface specifications and process descriptions, and acknowledge also the lack of informal coordination opportunities in distributed sites. These observations led to the development of an empirical theory of coordination for engineering decisions [Herbsleb and Mockus, 2003] which was later applied to identify coordination requirements among software developers that can be used to improve the design of collaboration tools [Cataldo et al., 2006].

In order to represent coordination and information flow in requirements engineering activities, Schneider et al. [2008] developed a notation that can be used to model both formal and informal communication. A benefit of the flow notation is that it can be used to describe the officially required and the actually executed process, showing differences and instances where improvements for coordination can be implemented [Stapel et al., 2007]. Furthermore, FLOW mapping has been used to plan and manage communication in distributed teams [Stapel et al., 2011].

## 2.2 Test Case Selection

Test case selection is a central activity in regression testing [Yoo and Harman, 2012]. In general, the goal is to choose a subset of the existing test cases because the execution of the whole set is impractical due to time or resource constraints. The problem of selecting a subset of test cases from a larger set is similar to test suite minimization. The decision in test case selection is however based on changes in the system under test rather than on an optimal functional coverage with a minimum amount of test cases [Yoo and Harman, 2012].

An abundance of techniques for test case selection exist, surveyed by Yoo and Harman [2012] and Engström et al. [2010]. One dimension on which these techniques can be classified is whether they use implementation information of the system under test or not. White box test case selection techniques exploit, for example, changes in source code, analyze data and control flows, and execution traces [Yoo and Harman, 2012]. The approach we develop and evaluate in Chapter 6 can be classified as a white-box technique since we use product configuration settings to trace source code to system test cases. Black box test case selection techniques are less common [Engström et al., 2010] and use, for example, changes in design documentation to perform impact analysis [Yoo and Harman, 2012]. The approach we develop and evaluate in Chapter 7 can be classified as black-box technique since we use product risk specifications as input for test

cases selection. Risk specifications provide an estimation of the products risk exposure [Amland, 2000], i.e. a combination of the likelihood that a particular feature contains a fault and the estimated cost of that fault appearing in production.

### 2.3 *Traceability Recovery*

The mapping of product functionality from one particular artifact to another, is related to traceability [Gotel and Finkelstein, 1994] in general, and to the concept assignment problem [Biggerstaff et al., 1993] in particular. Concept assignment is the process of mapping concepts from the problem domain (e.g. a requirement or feature expressed in potentially ambiguous and imprecise natural language) to the solution domain (e.g. an algorithm expressed in precise and unambiguous numerical computations). Research on traceability focused on tracing requirements to downstream artifacts (e.g. Gotel and Finkelstein [1994], Ramesh et al. [1997]) enables many software engineering activities such as regression testing and impact analysis [Gotel et al., 2012]. However, the cost of establishing and maintaining traceability manually reduces its adoption in industry [Spanoudakis and Zisman, 2004, Heindl and Biffl, 2005]. This has spawned research on techniques that aim at recovering traceability links by analyzing the content of project artifacts. Recovering traceability links between source code and natural language documentation using information retrieval (IR) techniques was pioneered by Antoniol et al. [1999, 2000], Maletic and Valluri [1999] and Maletic and Marcus [2000, 2001]. These early studies envisioned the potential of IR techniques to support software engineers in program comprehension [Maletic and Valluri, 1999, Antoniol et al., 1999], requirement tracing and impact analysis, software reuse and maintenance [Antoniol et al., 1999]. Borg et al. [2014] provide an extensive review of traceability recovery techniques.

## 3 RESEARCH QUESTIONS

A research question is an expression that formulates a scientific knowledge gap. Answering this question fills this gap, but most certainly leads also to new research questions. Formulating a set of research questions is the start of any scientific inquiry, the one leading to this thesis being no exception.

The research questions in this thesis are derived from two major research goals: (RG-I) Understand REST alignment and (RG-II) Improve REST alignment. The particular objectives of RG-I are to establish a common understanding between industry and research, but also among researchers, what characterizes REST alignment and what components are essential for its understanding. This forms the basis and is a prerequisite to address RG-II,

**Research Goal I: Understand REST alignment**

Research questions related to the industry relevance and the underlying principles of REST alignment.

Chapter  
2 3 4 5 6 7

**RQ-1** To what extent is REST alignment a challenge in industry?

*RQ-1.1* What are the current challenges in achieving REST alignment?

*RQ-1.2* What are the practices that support REST alignment?

*RQ-1.3* Which industry practices address REST alignment challenges?

**RQ-2** How can REST alignment be characterized?

*RQ-2.1* What are the underlying principles of REST alignment?

*RQ-2.2* How can practices or techniques supporting REST alignment be characterized?

*RQ-2.3* How can REST alignment be assessed?

**Research Goal II: Improve REST alignment**

Research questions related to assessing, improving and supporting REST alignment in industry.

Chapter  
2 3 4 5 6 7

**RQ-3** Which method can be used to determine the state of REST alignment in a software development project?

*RQ-3.1* To what extent are the underlying REST principles useful to elicit improvement opportunities?

*RQ-3.2* To what extent does the defined REST assessment method depend on the particular context it is used?

**RQ-4** To what extent can test cases replace requirements?

*RQ-4.1* Why do test cases replace requirements in industry?

*RQ-4.2* How can test cases replace requirements in industry?

**RQ-5** To what extent can information retrieval (IR) techniques support REST alignment?

*RQ-5.1* To what extent can state-of-the-art IR techniques be applied in a large-scale industrial context?

*RQ-5.2* To what extent do the used software development artifacts influence the performance of IR techniques?

*RQ-5.3* To what extent can the studied IR techniques support test case selection?

Figure 1.2: Research questions answered in this thesis. A ● indicates the chapter in which the question is answered. A ○ indicates related chapters.

with the objective to assess and improve REST alignment in varying industry contexts. Note that both research goals are driven by the agenda of the research project (see the section on funding in the preface of this thesis) and the goals and requirements of the companies participating in that project (see Section 4.3 where we illustrate the research setting).

Figure 1.2 illustrates the five main research questions connected to RG-I and RG-II, driving the content in this thesis. Each research question is associated with one or more chapters where the question is answered (indicated by a ● in Figure 1.2). Note however that broader research questions connected to RG-I, Understand REST alignment, have related chapters throughout the thesis, indicated by a ○ in Figure 1.2. In the remainder of this section, we discuss the research questions and their relationships in more detail.

### 3.1 RG-I: Understand REST Alignment

With understanding we mean the ability to describe a phenomenon in different contexts, eventually abstracting the observed entities in such a way that they can be communicated, leading to the generation of new ideas and research. There are two main research questions and six subquestions associated with RG-I. We begin with a question that anchors our inquiry in the state-of-practice, ensuring industrial relevance of the research that follows. RQ-1 asks *to what extent is REST alignment a challenge in industry?* We aim to identify the particular challenges faced by industry, but also practices they apply or intend to apply to improve REST alignment. RQ-1 and its subquestions are answered in Chapter 2. Furthermore, current industry challenges in achieving REST alignment (RQ-1.1) are also discussed in Chapter 4 (in five different companies), and in Chapters 6 and 7 (focus is one particular company where we also collaborated to identify solutions). Practices supporting REST alignment (RQ-1.2), or the lack thereof, is a central theme throughout this thesis, hence discussed in each chapter. Solutions observed in industry addressing particular REST alignment challenges (RQ-1.3) are also prevalent and hence discussed in Chapters 4, 5, 6 and 7 of this thesis.

While RQ-1 is in line with the empirical principles of this thesis, its answers are confined by the context of the companies in which they are set (see Table 1.2 in Section 4.3 which illustrates that “industry” refers only to the involved companies in this thesis, not industry as a whole). Therefore, we investigate with RQ-2, *How can REST alignment be characterized?*, and its subquestions how to distill a set of principles of REST alignment practices that originate from research. This is the central theme of Chapter 3, while REST alignment is also characterized to some extent in Chapters 2 and 4. The underlying principles of REST alignment in general (RQ-2.1) and of

particular practices (RQ-2.2) are identified in Chapter 3 and validated in Chapter 4. The assessment of REST alignment (RQ-2.3) is central to both Chapter 3 and 4.

### 3.2 RG-II: Improve REST Alignment

RG-I and RG-II are inherently connected: the latter, improvement, builds upon the former, understanding. All three main research questions associated with RG-II aim at studying means to support REST alignment. While we investigate three orthogonal strategies, the first based on assessment (Chapter 4), the second on reuse (Chapter 5), and the third on re-engineering (Chapters 6 and 7), they share the idea that coordination through information is central for achieving REST alignment (and for coordination in general [van de Ven et al., 1976]), a principle that has been established in Chapter 3.

With RQ-3, *Which method can be used to determine the state of REST alignment in a software development project?*, we follow the objective of validating the identified principles from RG-I by putting them into practice, i. e. using them as a mean to assess REST alignment and to identify improvement opportunities. This question (RQ-3.1) is answered in Chapter 4 and to some extent in Chapter 3. Then we ask whether the assessment approach, and the underlying principles, are general, that is, can be applied in a wide variety of contexts (RQ-3.2). This is answered in Chapter 4.

With RQ-4, *To what extent can test cases replace requirements?*, we take a closer look at reusing test cases as requirements specifications, a practice we observed in the companies studied in Chapter 2. In particular, we ask why (RQ-4.1) and how (RQ-4.2) this practice is applied, answering these questions in Chapter 5.

With RQ-5, *To what extent can information retrieval techniques support REST alignment?*, we steer our investigation into supporting industry in test case selection in a software product line context. Test case selection is relevant in the context of REST alignment as efficient and effective quality assurance depends on the ability to base decisions upon information that originates from requirement and product management. We explore two approaches:

- In Chapter 6 we follow a white-box approach, aiming at recovering traceability between the source code of a particular product variant and system test cases, and thereby providing support for test case selection. We answer questions regarding the scalability (RQ-5.1), difficulty (RQ-5.2) and effectiveness (RQ-5.3) of this approach.
- In Chapter 7 we explore a black-box approach, where the decision support is based on project documentation rather than product artifacts. We answer again questions regarding scalability (RQ-5.1) and

|   |   |
|---|---|
| <b>Post-positivism</b>  | <b>Constructivism</b>   |
| <ul style="list-style-type: none"> <li>- Determination</li> <li>- Reductionism</li> <li>- Empirical observation</li> <li>- Theory verification</li> </ul> | <ul style="list-style-type: none"> <li>- Understanding</li> <li>- Multiple participant meanings</li> <li>- Social and historical construction</li> <li>- Theory generation</li> </ul> |
| <b>Transformative</b>   | <b>Pragmatism</b>   |
| <ul style="list-style-type: none"> <li>- Political</li> <li>- Power and justice oriented</li> <li>- Collaborative</li> <li>- Change-oriented</li> </ul>   | <ul style="list-style-type: none"> <li>- Consequences of actions</li> <li>- Problem-centered</li> <li>- Pluralistic</li> <li>- Real-world practice oriented</li> </ul>                |

Figure 1.3: Research worldviews (adapted from Creswell [2013])

effectiveness (RQ-5.3) of this approach. Note that RQ-5.2 is not relevant for a black-box approach to test case selection.

Next we discuss how we answered the stated research questions.

#### 4 RESEARCH APPROACH

In this section, we provide first some motivation for the chosen research approach and discuss advantages and disadvantages of the empirical methods typically used in software engineering research [Easterbrook et al., 2008] in Section 4.1. Then we introduce the research framework used to answer the research questions (Section 4.2), describe the research setting (Section 4.3), and conclude with the limitations of this thesis (Section 4.4).

Before we discuss empirical research methods, we first shed light on the research theoretical perspective that guided the scientific inquiry in this thesis. Research theory roughly distinguishes between four worldviews [Creswell, 2013]: post-positivism, constructivism, transformative, and pragmatism. The characteristics of these views are summarized in Figure 1.3. The *post-positivist*<sup>3</sup> worldview is often associated with the traditional scientific method where a researcher starts with a theory, collects and analyses data to either support or reject the theory, refining the theory along this process [Creswell, 2013]. Post-positivism favors quantitative research designs such as experiments [Wohlin et al., 2000] and surveys [Robson, 2002]. On the other hand, the *constructivist* worldview is typically associated with qualitative research designs [Creswell, 2013], such as case studies [Yin, 2003], grounded theory [Glaser et al., 1968] and ethnographies

<sup>3</sup> Post-positivists acknowledge, in contrast to positivist, the influence of the researcher on the studied phenomenon, which can only be known probabilistically due to the researchers' limitations [Robson, 2002].

[Robson, 2002]. In this view, researchers construct the meaning of a situation, in a particular context, from multiple and varied observations; rather than starting from a theory, researchers generate under the influence of their own interpretation a pattern or theory that describes a phenomenon [Creswell, 2013]. In the *transformative* worldview, research strives at changing the studied phenomenon itself and/or the people or institutions participating in the inquiry [Creswell, 2013]. This view is associated with quantitative, qualitative and mixed-method research designs, while the participants are actively involved in research design [Mertens, 2005]. Action research [Brydon-Miller et al., 2003] is closely associated with a transformative worldview [Easterbrook et al., 2008]. Researchers with a *pragmatic* worldview focus on the research problem and utilize all approaches available to solve the problem [Rossman and Wilson, 1985]. A pragmatist researcher chooses the *what* and the *how* to research based on the intended consequences [Creswell, 2013], adopting an engineering approach to research [Easterbrook et al., 2008].

In this thesis, we have adopted a *pragmatic* worldview and conduct research to solve problems. This does however not mean that we “invent” problems and solve them in the seclusion of our offices. As pragmatic researchers we study the phenomenon of REST alignment in industry and research (RG-I), performing real world inquiries [Robson, 2002]. Then we develop solutions to assess and support REST alignment (RG-II).

#### 4.1 Research Methods in General

A research method describes the process by which a knowledge gap can be closed. This process can be rationalist, based on pure reason and a-priori knowledge, or empirical, based on evidence originating from data. Rationalism promotes the idea that knowledge is part of our rational nature or can be gained by logical deduction, without sensory experience [Markie, 2015]. Empiricism, on the other hand, promotes the idea that all knowledge in a subject originates from sensory experience [Markie, 2015], i. e. data sampled from the world that surrounds us and analyzed to provide evidence for or against a knowledge claim. Such data can be either of quantitative, i. e. numbers, or of qualitative nature, e. g. text or audio-visual data [Runeson and Höst, 2009]. While research in theoretical computer science, due to its close association with mathematics and logic, often follows a rationalist paradigm [Eden, 2007], research in software engineering tends to an empirical, evidence-based approach [Kitchenham et al., 2004, Eden, 2007, Shull et al., 2008].

The selection of a particular research method is influenced by the research worldview one adopts [Easterbrook et al., 2008] and the knowledge gap that needs to be closed, i. e. the purpose of the research [Robson, 2002,

Table 1.1: Characteristics of empirical software engineering research methods

|           | Experiment      | Case study  | Survey          | Ethnography    | Action research |
|-----------|-----------------|-------------|-----------------|----------------|-----------------|
| Purpose   | Explanatory     | Exploratory | Descriptive     | Descriptive    | Improving       |
| Data      | Quantitative    | Both        | Both            | Qualitative    | Qualitative     |
| Worldview | Post-positivist | All four    | Post-positivist | Constructivist | All four        |

p. 80]. Different research methods serve different purposes [Runeson et al., 2012, p. 13]:

- Exploratory research seeks new insights, generates ideas and hypotheses.
- Descriptive research illustrates contemporary phenomena.
- Explanatory research seeks explanations for a phenomenon.
- Improving research seeks to positively change a certain aspect of a phenomenon.

Easterbrook et al. [2008] discuss the five most relevant empirical research methods for software engineering: experiment, case study, survey, ethnography and action research. These methods serve different research purposes and have different flaws, leading to validity threats that limit the strength of evidence that can be generated on the studied subject [Easterbrook et al., 2008]. We briefly discuss these empirical research methods for software engineering and their liabilities, while more in-depth analyses and examples are provided in the referenced literature. Table 1.1, adapted from Runeson et al. [2012], Wohlin et al. [2000] and Easterbrook et al. [2008], summarizes empirical research method characteristics.

EXPERIMENTS manipulate one or more independent variables, the treatment, to measure the effect on one or more dependent variables [Easterbrook et al., 2008], that is, the aim is to evaluate our belief on the existence of particular cause-effect relationships [Wohlin et al., 2000, p. 31]. Knowledge is gained by testing whether a hypothesis, defined by the involved variables and often based on a particular theory, can be refuted, given the collected evidence. Experiments are closely associated to the (post)-positivist worldview [Easterbrook et al., 2008]. Experiments can be performed on human subjects [Easterbrook et al., 2008], where the treatment could be the use of a particular tool or practice, or on methods [Hayes et al., 2006], where the treatment are particular parameters of the technique under study. Treatments are assigned randomly to subjects in order to avoid systematic bias. If this is not possible, e.g. due to ethical reasons or resource constraints in the company where the field experiment is conducted, we speak of quasi-experiments [Wohlin et al., 2000, Easterbrook

et al., 2008]. The data collected in experiments is typically of quantitative nature, leading to forms of analysis that are quantitative, i.e. descriptive and inferential statistical analysis.

The strength of experiments is that they allow the researcher to define an experimental design [Wohlin et al., 2000] that matches precisely the knowledge gap, provided that confounding variables [Pearl, 1998, Wohlin et al., 2000] are known and controlled for. This is at the same time a weakness of experiments, since it forces the researcher to decide in advance which variables are important to manipulate and observe, and which are irrelevant [Easterbrook et al., 2008].

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” [Yin, 2003, p. 13]. An important distinguishing feature between experiments and case studies is the level of control the researcher has on the variables of interest [Yin, 2003, p. 7]. The design and implementation of a case study is still guided by one or more research questions; however, the collected data originates from purposive rather than random sampling [Easterbrook et al., 2008]. In other words, experiments sample over the variables being manipulated, while case studies sample from variables representing a typical situation [Wohlin et al., 2000, p. 12]. Hence, case study research fits into contexts where there are more variables of interest than data points and multiple sources of evidence are available [Yin, 2003, p. 14]. While case studies are generally associated with qualitative data [Easterbrook et al., 2008], Yin [2003, p.14] points out that cases studies can include or even be limited to quantitative data. When the collected data is of qualitative nature, it is important to define and follow a rigorous analysis procedure that enables traceability from the raw data, intermediate reduced data that allows easier navigation, to the final conclusions. The goal of any qualitative data analysis is to maintain a chain of evidence that allows the reader to follow the trace of evidence from the data to the derived conclusions [Runeson and Höst, 2009]. Case study research is associated with all four research worldviews, even though the choice of data collection and analysis methods varies in each viewpoint [Easterbrook et al., 2008].

The strength of case studies is the embracement of “rich” evidence, originating from qualitative data, potentially creating an understanding that mirrors reality by considering a multitude of variables, as opposed to the reductionism of experiments [Easterbrook et al., 2008]. On the other hand, it is more difficult to ensure the rigorousness of qualitative data collection and analysis methods [Easterbrook et al., 2008], which depends on the expertise of the researcher [Robson, 2002, p. 457].

**SURVEY RESEARCH** is used to characterize a population by drawing a representative sample from which data is collected [Easterbrook et al., 2008], typically by closed-ended questions [Robson, 2002, p. 234]. The goal is to generalize from the characteristics of sample to the population [Easterbrook et al., 2008]. The design of the data collection instrument, typically a self-administered questionnaire even though interviews are also possible, and the sample selection are crucial for the validity of the survey results [Robson, 2002, p. 231]. Survey research is strongly associated with the post-positivist worldview [Easterbrook et al., 2008].

The strength of surveys is that they can be administered at a relatively low cost, while still providing large amount of data, that can be generalized to the targeted population [Robson, 2002, p. 233]. On the other hand, sampling and instrumentation biases are difficult to control [Easterbrook et al., 2008, Robson, 2002, p. 233].

**ETHNOGRAPHIES** are typically employed when the social and technical interaction of certain communities or groups of people are the focus of the research [Easterbrook et al., 2008]. Central differences of ethnographies to the previously discussed methods are the immersion of the researcher with the studied environment and the deliberate omission of ideas or concepts at the beginning of the study and when analyzing the collected data [Robson, 2002, p. 186]. These features put ethnographies close to the constructivist worldview [Easterbrook et al., 2008].

Ethnographies are particularly valuable when a new or different field is focus of the research interest as it allows to acquire a deep understanding guiding follow-up studies [Robson, 2002, p. 190]. On the other hand, ethnographies require a long-term commitment from both the researcher and the researched environment (in the order of magnitude of years) [Robson, 2002, p. 187], and raise the challenge of observation, data collection and analysis without biasing preconceptions [Easterbrook et al., 2008].

**ACTION RESEARCH** aims at “solving organizational problems through intervention while at the same time contributing to scholarly knowledge” [Davison et al., 2004]. The quality of action research depends on several criteria: the relevance of the original problem that needs solving, the relevance of the knowledge outcome for the participants, and the knowledge gain resulting from the critical reflection on the implemented changes [Easterbrook et al., 2008]. The researcher needs thereby to balance the needs of the client to solve a particular problem and the requirements that enable a scientific dissemination of the acquired findings [Davison et al., 2004]. The collaborative and changed-focused nature of action research associates it strongly with the transformative worldview [Easterbrook et al., 2008, Robson, 2002, p. 215]. However, shifting the focus to a comparison between the state before and after the change puts action research into the

post-positivist territory, while focusing on the perceptions of the change process associates actions research with a constructivist worldview [Easterbrook et al., 2008].

The strength of action research is its potential to deliver both relevant and rigorous [Ivarsson and Gorschek, 2011] research results. While lack of rigor has been criticized in the past [Easterbrook et al., 2008], there exist guidelines, e.g. Davison et al. [2004], that provide new researchers a framework to design action research studies. A liability of action research is the researchers' potential loss of power to the investigated organization in design and data collection decisions [Robson, 2002, p. 216].

**MIXED METHODS DESIGNS.** As illustrated in the previous paragraphs, each research method exhibits certain strengths but also weaknesses. The premise of mixed methods designs is that by combination, the strength of one method may compensate for the weakness of another [Creswell, 2013, p. 15]. The question is however how to combine the methods to achieve the desired effects. There exist three broad strategies [Creswell, 2013, pp. 15–16]:

- In *convergent parallel mixed methods*, the researcher collects quantitative as well as qualitative data, usually in the same time-frame, and integrates both sources of information in the analysis to answer the knowledge gap. Note that methodological triangulation promoted by case study research [Runeson and Höst, 2009] can therefore be seen as a form of this mixed method strategy.
- In *explanatory sequential mixed methods*, the researcher commences with a quantitative study which is then complemented by a qualitative study that aims to explain the quantitative results.
- In *exploratory sequential mixed methods*, the researcher commences with a qualitative study to explore the area of interested and to identify important variables. This information is then used to support the definition of a quantitative study.

While mixed method research can be conducted with any research philosophy, it is often associated with a pragmatic worldview [Easterbrook et al., 2008].

#### 4.2 Research Framework in this Thesis

We put now the studies presented in Chapter 2–7 into a research framework that illustrates how we addressed RG-I and RG-II. While this placement is a post-fact, static depiction of a research process that developed over time, the following principles were adhered to throughout design and implementation of theses studies:

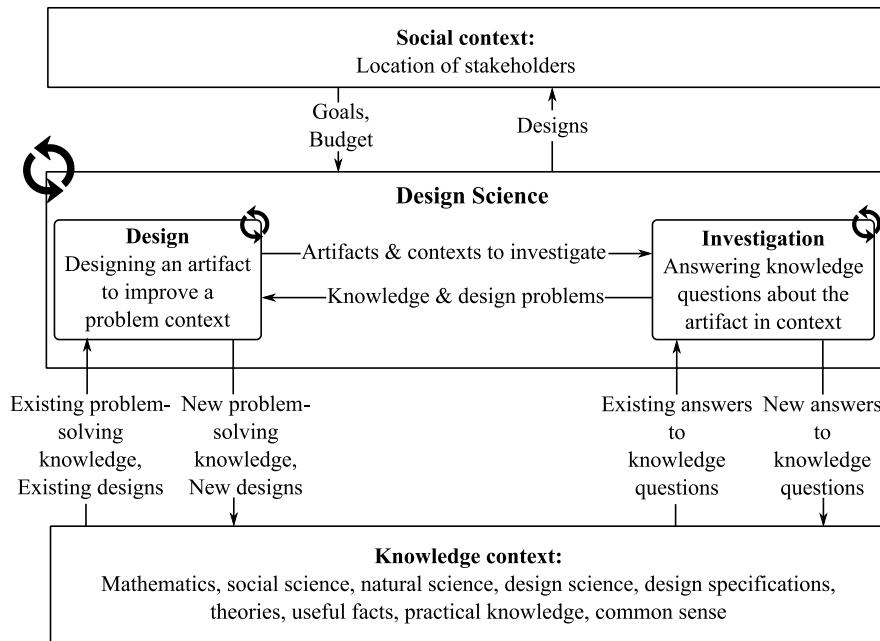


Figure 1.4: Design science framework (adapted from Wieringa [2014a])

- Problem identification guided by empirical evidence
- Industry-relevant solutions that scale to real-world problems
- Solution development drives knowledge gain and vice versa
- Iterative improvement of solutions

These principles map well into the design science [March and Smith, 1995, Hevner et al., 2004, Wieringa, 2014a] paradigm which we will use henceforth as our reference research framework (Figure 1.4). Design science builds and evaluates, constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices) and instantiations (implemented and prototype systems) that serve human purposes [March and Smith, 1995, Hevner et al., 2004]. Central to design science research is that understanding of a problem and its solution is gained by building and application of an artifact within context [Hevner et al., 2004, Wieringa, 2014a, p. 3]. The artifacts developed in design science are typically addressing a class of problems while targeted at the specific problem of a specific client [Iivari and Venable, 2009]. The design science research process is driven by two cycles: (1) find solutions for design problems, calling for a change in the real world and based on the requirements of stakeholders goals (design and engineering cycle); (2) an-

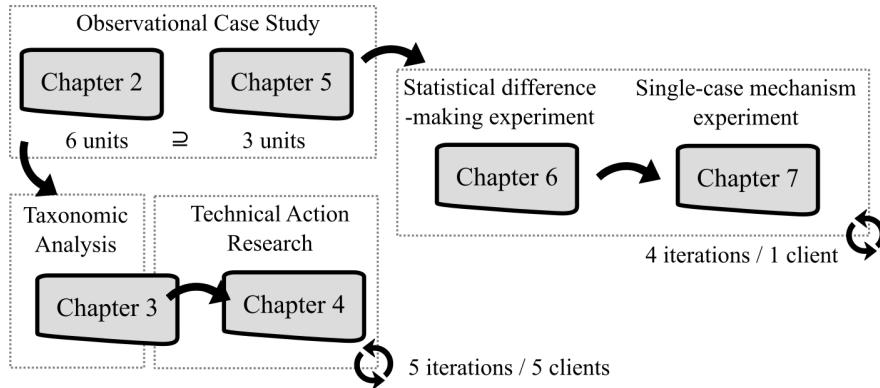


Figure 1.5: Research framework connecting the chapters in this thesis

swering scientific knowledge questions on the artifact in context (empirical cycle) [Wieringa, 2014a, p. 4].

In the remainder of this section we discuss how we answered our knowledge questions (see Figure 1.2), drawing on the traditional research methods discussed in Section 4.1, and embedded in the context of the design science framework. Figure 1.5 provides an overview of the research framework connecting the chapters in this thesis. In the following paragraphs we discuss and motivate the taken research method choices.

The purpose of the study in Chapter 2 was exploratory (see RQ-1 in Figure 1.2), leading to the choice of an observational case study to perform problem investigation [Wieringa, 2014a, p. 225]. One could argue that an ethnographic study would have been a valid alternative approach to fill this knowledge gap, given the underdeveloped research area (overlap between RE and ST) and the few preconceptions the involved researchers had (simple conceptual model, input from industry). While it probably would have been possible to acquire the resources to implement long term ethnographic studies in each of the six participating companies, it would also have been very difficult to coordinate the likely inhomogeneous data collection and analysis among the involved researchers. The relatively short data collection period (still spanning over 12 months) was supported by guidelines that, while allowing for exploration, focused the data elicitation (semi-structured interviews) to the phenomenon of interest, i. e. REST alignment. Chapter 5 draws also from the data collected in this observational case study. This was made possible by a three stage data analysis process where the first stage consisted of assigning codes to the collected data. These codes allowed for the navigation and filtering of the data, depending on the stated knowledge questions. While for the research questions in Chapter 2 all six companies provided relevant data, the research

questions stated in the study in Chapter 5 (see Figure 1.2) produced data from a subset of the studied companies. The results from the observational case study provided input and motivation for the remaining studies, as shown in Figure 1.5 and discussed next.

While the purpose of the study in Chapter 3 was also exploratory, a major difference to Chapter 2 is the focus on developing a conceptual framework that can be used to describe and communicate the observed phenomenon [Wieringa, 2014a, p. 73]. A conceptual framework provides an analytical abstraction that captures defining and useful dimensions of the observed, empirical phenomenon (REST alignment). We developed a taxonomy by analyzing a set of published REST alignment methods. This process was iterative and inductive, which complies to empirical principles of knowledge acquisition [Markie, 2015]. A viable alternative to this approach would have been to conduct a grounded theory study [Robson, 2002, pp. 190–193]. Indeed, there are parallels in the methods we used to identify and validate the taxonomy dimensions (purposive sampling, open coding, constant comparison) and grounded theory techniques. In order to operationalize the taxonomy, we developed and piloted an assessment method, REST-bench, that should be seen in the context of the research method applied in Chapter 4, discussed next. Note that with the conclusion of Chapter 3, we turn now to research goal RG-II, improve REST alignment, which changes also the focus of the research methods to explanatory/improving approaches.

The purpose of the study in Chapter 4 was on one hand improving, i.e. to assess and potentially induce change in the studied REST alignment phenomenon, and on the other hand to find support for the conceptual framework developed in Chapter 3. The challenge of these two objectives is that they exhibit, to some extent, conflicting requirements, rendering the choice of research method difficult. The improving purpose would probably be best served with an action research approach which requires a prolonged involvement in a certain environment (e.g. a software development project) [Petersen et al., 2014]. However, to validate the theoretical perspective of the REST taxonomy, a varying environment would be preferable since that would provide indications of the taxonomy's stability and usefulness in different situations. We chose in the study presented in Chapter 4 to apply technical action research (TAR) [Wieringa and Morali, 2012] which consists of using an experimental artifact, REST-bench in this case, to help a client and, at the same time, learn about its effects in uncontrolled practice [Wieringa, 2014a, p. 269]. We applied REST-bench at five diverse clients, iteratively improving the method and at the same time helping the client to find improvement opportunities for REST alignment. However, we traded depth of the improvement cycle typically encountered in action research against breadth of validation. Concretely, this means that we applied REST-bench in each case company to identify improvement

opportunities and confirmed their relevance with the study participants, did however not implement, monitor and evaluate the proposed improvements (depth trade-off). On the other hand, we applied REST-bench in five different settings and integrated lessons learned into the approach (breadth trade-off). This allowed us to answer the stated knowledge questions RQ-3.1 and RQ-3.2 in Figure 1.2.

In Chapters 6 and 7 we turned the strategy from Chapter 4 around and focused on validation depth instead of breadth. This allowed us to study whether a general solution (traceability recovery) can be used to solve a REST alignment challenge, described in Chapters 2 and 5, encountered at one particular client. The purpose of the study in Chapter 6 was explanatory at the level of each experimental setup where we studied the effects of varying information retrieval (IR) techniques on the performance of the task at hand. However, looking at the study as a series of three experiments, the purpose was to explore different setups, learning from their implementation and evaluation, and eventually to identify a configuration of IR techniques that can indeed provide support for the client's problem. Within the design science framework, this research method can be classified as a difference-making experiment [Wieringa, 2014b], even though the non-random assignment of treatments to objects qualifies the design as a quasi-experiment [Wohlin et al., 2000] of methods [Hayes et al., 2006]. The purpose of the study in Chapter 7, while continuing the solution development in collaboration with the client and addressing similar research questions (see RQ-5 in Figure 1.2) as in Chapter 6, was also explanatory, i. e. to provide a proof-of-concept implementation for a test case selection decision support system. The goal of a single-case mechanism experiment is to describe and explain cause-effect behavior [Wieringa, 2014b], which is in this particular study the effect of existing test case categorizations on the quality of test case selections. Together with the experiments in Chapter 6, the single-case mechanism experiment in Chapter 7 form an in-depth investigation, over four iterations at one client, on the potential of IR techniques for test case selection support.

#### 4.3 Research Setting and Data Collection

Each chapter in this thesis is firmly grounded in empiricism and connected to one or more companies that contributed to this research with data, collaboration and experience. Table 1.2 provides an overview on the involvement of the companies in each chapter and which type of data (qualitative and/or quantitative) was collected. Note that company names A–F are anonymized for confidentiality reasons.

Chapter 2 reports on a case study including six companies (A–F) where in total 30 practitioners were interviewed. The evidence collection was

Table 1.2: Companies contributing to each chapter of this thesis, indicating also type of collected data: qualitative (QL) and quantitative (QN).

| Company            | Chapter |                |       |                |    |    |
|--------------------|---------|----------------|-------|----------------|----|----|
|                    | 2       | 3 <sup>1</sup> | 4     | 5 <sup>2</sup> | 6  | 7  |
| A                  | QL      |                |       | QL             | QN | QN |
| B                  | QL      |                |       | QL             |    |    |
| C                  | QL      |                |       |                |    |    |
| D                  | QL      |                |       |                |    |    |
| E                  | QL      | QL/QN          | QL/QN |                |    |    |
| F                  | QL      |                |       | QL             |    |    |
| CompuGroup Medical |         |                | QL/QN |                |    |    |
| ST Ericsson        |         |                | QL/QN |                |    |    |
| Telenor            |         |                | QL/QN |                |    |    |
| Volvo Cars         |         |                | QL/QN |                |    |    |

<sup>1</sup> The data collected in the pilot study at Company E was reused in Chapter 4.

<sup>2</sup> The data originates from the interviews conducted in the case study reported in Chapter 2.

guided by a semi-structured interview. Interviews were audio recorded and then transcribed, resulting in 300 pages qualitative data expressing experiences, opinions, analyses and generally answers to the questions stated by the interviewers. Note that a subset of the collected data was reused in Chapter 5.

The data collected in Chapter 3 and Chapter 4 was elicited at two points in time and served two different purposes. Qualitative data originated from the assessments performed with REST-bench, and was elicited with a structured interview guide. After the assessment, participants rated the method via a questionnaire that elicited qualitative and quantitative data.

Both in Chapter 6 and Chapter 7, we collected textual data that was converted into term-document matrices, which are the basis for the implementation of the evaluated IR models. In Chapter 6, further quantitative data was collected from the performance of the evaluated IR configurations and from test engineers who rated test case rankings. In Chapter 7, we quantitatively evaluated topic models to perform model selection, collected information on historic test runs, and elicited data from a test engineer who rated the relevance of the test case selection performed with the developed method.

#### 4.4 Limitations

One reason for adhering to a particular research method is that each has well known inherent weaknesses that affect the study's findings validity. Commonly used categorizations of threats to validity in empirical software engineering research origin from Wohlin et al. [2000] and Runeson and Höst [2009]. While some of these threats can be outbalanced by certain countermeasures, some might need to be prioritized since they are mutually affecting each other [Wohlin et al., 2000] and some might need to be accepted due to time/resource limitations of the research project.

Since we discuss validity threats and taken countermeasures in each chapter, we focus in this section on pointing out the major limitations that nevertheless exist and affect the thesis as a whole. The data collection and analysis process in the observational case study in Chapter 2 required the collaboration of several researchers to be feasible within acceptable time. While this introduced observer triangulation in the data collection and reduced potential bias in the data analysis, it likely reduced also internal consistency in both processes. Evidence for this can be found in Chapter 2 where the identified alignment practice of using test cases as requirements was associated with company A and F. The re-analysis of the data, with the particular focus on this alignment practice in the study presented in Chapter 5, has shown that Company B uses this practice quite prominently as well. Potential lack of internal consistency in the data interpretation is therefore one limitation of the studies in Chapter 2 and 5.

The study in Chapter 3 suffers from a similar limitation as systematic literature reviews performed with a snowball sampling search strategy [Wohlin, 2014]: the starting set of papers influences the direction into which new samples are collected and analyzed. While this is not directly transferable to the taxonomy creation process we applied, a bias towards studies focusing on artifacts rather than, for example, processes and people would influence the direction of the data extraction and abstraction process that led to the identified dimensions. As such, the REST taxonomy is one possible, but not the exclusive way to characterize REST alignment, and one alternative is the theory proposed by Bjarnason et al. [2014b].

REST-bench, illustrated in Chapter 4, has not been independently validated. The author of this thesis has in all five cases applied the assessment framework alone, such that his expertise, or lack thereof, confounds the conclusions on the usefulness of the approach. This is a limitation that can be addressed by releasing the solution [Gorschek et al., 2006] to other researchers and/or practitioners.

The main limitation of the experimental setups in Chapter 6 is the low number of the samples on which we evaluated the IR technique configurations. This was a necessary compromise to be able to study the question whether state-of-the-art IR techniques are applicable on industry-grade

data. Collecting, understanding and preparing this data for an experiment of this complexity, where both feasibility and effectiveness of the approach were evaluated, turned out more challenging than expected.

RiTTM, illustrated in Chapter 7, has seen limited validation within the context of Company A. While we do not expect a systematic bias in the selection of the studied objects, RiTTM has been applied by the author of this thesis and the test engineers at Company A only evaluated the results. As such, RiTTM is at its current state a proof-of-concept that still requires dynamic validation [Gorschek et al., 2006] in industry.

## 5 OVERVIEW OF THE CHAPTERS

This section provides a brief overview of each chapter in this thesis.

### 5.1 *Chapter 2*

The premise of this chapter (and of the thesis as a whole) is that the alignment/coordination of requirements engineering and software testing activities/practices/tools/artifacts leads to synergies that provide benefits that go beyond improving the individual areas alone. This assumption is to some extent supported in literature, e.g. by Regnell et al. [2000], Graham [2002], Cheng and Atlee [2007], Bertolino [2007], Martin and Melnik [2008], Uusitalo et al. [2008], Post et al. [2009], Mäntylä and Itkonen [2014] and Merz et al. [2015].

The main question in this chapter is the extent to what REST alignment is a challenge in industry (RQ-1). We investigate this question in a case study involving six companies, interviewing 30 practitioners that exhibit a wide range of expertise not only in requirements engineering and software testing, but also in product/project/process management and software development. The nature of this inquiry is exploratory [Robson, 2002], therefore not guided by a particular framework or theory except a rough image on RE and ST relationships (see Figure 2.2). The contributions of this investigation in industry can be found on multiple levels:

**LEVEL 1** The identification of concrete and relevant barriers in supporting, establishing or maintaining REST alignment, providing evidence that addressing them might be beneficial.

**LEVEL 2** The bootstrap of further research, that focuses on addressing particular challenges identified in industry.

**LEVEL 3** Establishment of a research community that is cross-fertilized by ideas originating from both RE and ST areas, and from the technical and human-oriented aspects of software engineering.

Direct contributions of the case study described in Chapter 2, relating to Level 1, is evidence and characterization of 16 REST alignment challenges, categorized in four classes (requirements specification quality, software testing quality, requirements abstraction levels, traceability), 27 practices in six categories (requirements, validation, verification, change, tracking, tools) and a proposed mapping between challenges and practices addressing them.

An indirect contribution of Chapter 2 is the further research it has spawned (Level 2), which can be found in this thesis, but also in the work of two other researchers involved in the case study (Bjarnason [2013], Borg [2015]). The research in Chapter 3 is partly motivated by the lack of a theoretical lens [Easterbrook et al., 2008] that could have guided both data elicitation and analysis in the case study presented in Chapter 2. Further research in Chapters 6 and 7 investigates solutions for the particular REST alignment challenges of case company A, first studied in Chapter 2 and then in more depth in Chapter 5.

Another indirect contribution of the work in Chapter 2 is the connection of researchers from different fields, leading to the organization and shaping of a dedicated workshop<sup>4</sup>, where the focal point is the exchange of ideas from both the RE and ST research communities (Level 3).

**MAIN FINDINGS.** Many REST alignment challenges but also practices are related to human and organizational sides of software development. While not a surprising result in itself, this new evidence provides one more data point that can support design decisions for REST alignment solutions. Requirements quality (verifiable, unambiguous, complete, up-to-date, specified at the needed level of detail) is central for any coordination within and across software development activities. Furthermore, company size, domain and product type influence the encountered REST alignment challenges, resulting in different incentives for addressing them, and leading to varying REST alignment practices.

## 5.2 *Chapter 3*

Together with Chapter 2, the research in this chapter furthers our understanding of REST alignment. While the case study in the previous chapter provides a realistic picture of the state-of-practice, the aim of Chapter 3 is to identify the fundamental components through which REST alignment can be characterized (RQ-2). The premise of this idea is that we can use these components to analyze, improve and even design REST alignment practices and techniques, but also to analyze the state of REST alignment, for example in a development project.

---

<sup>4</sup> International Workshop on Requirements Engineering and Testing, <http://ret.cs.lth.se/>

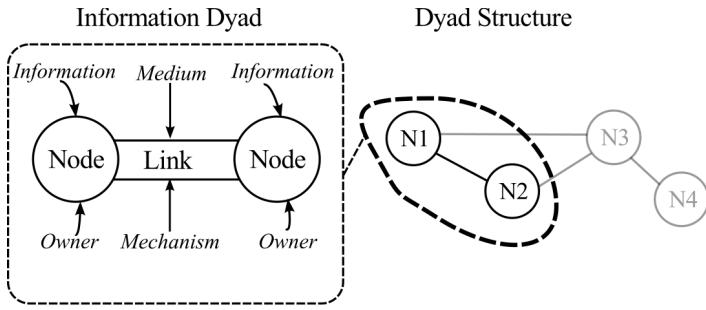


Figure 1.6: Information dyad and dyad structure

When we discuss alignment, not only in the context of RE and ST, we refer to the idea of coordinated functioning, that is, two or more entities (persons, teams) work together to achieve a common goal. Fundamental for any form of coordination is the exchange of information [van de Ven et al., 1976]. Therefore, the main contribution of Chapter 3 is the definition of the “information dyad”, the base construct we use to characterize REST alignment.

An information dyad has three components (see Figure 1.6): two nodes and link through which the nodes are connected. Nodes are characterized by the information they represent and the owner of that information. Links are characterized by the medium and mechanism through which information is aligned. An alignment method can consist of one or more information dyads, forming a dyad structure which exhibits certain properties (Table 1.3). We use information dyad characteristics, dyad structures and their properties to characterize not only REST alignment method but also the state of alignment.

An example for a REST alignment method is the approach proposed by Güldali et al. [2011] for deriving acceptance test plans from requirements specifications<sup>5</sup>. Their approach addresses inconsistencies, redundancies and dependencies originating from requirements that represent different viewpoints of a system which lead to complex and erroneous test plans. We identified two information dyads in their approach, linking requirements specified in natural language, requirements clusters and abstract test cases.

In total, we classified thirteen REST alignment methods published in literature, characterizing them by the information dyad construct and the dyad structure properties, allowing us to discuss the methods’ complexity and focus. Overall, we have observed a median of three information dyads per method, and a tendency to more nodes in the RE than in the ST phases.

---

<sup>5</sup> Details to this methods’ classification are provided in Chapter 3.

Table 1.3: Dyad structure properties

- 
- |                      |  |
|----------------------|--|
| <i>P<sub>1</sub></i> | Number of nodes – links between nodes need to be maintained over time. Hence, the total number of nodes allows one to reason on complexity and effort to maintain REST alignment.  |
| <i>P<sub>2</sub></i> | Branches – a branch exists, if a node acts as a source or sink for more than one node. Branches may reduce local complexity, require however also means to synchronize and merge information.  |
| <i>P<sub>3</sub></i> | Intermediate nodes – characterized by information that belongs to the design/analysis or implementation phases.  |
| <i>P<sub>4</sub></i> | RE and ST node proportion – assuming that a node is associated with a certain cost (e.g. establishing/maintaining the information therein and links in between), it is of interest to know the node distribution among the RE and ST phases. |
| <i>P<sub>5</sub></i> | Links – the linking mechanism between phases determines how changes of information are propagated.   |
| <i>P<sub>6</sub></i> | Scope – allows one to reason upon the interface between RE and ST and other phases.  |
- 

Assuming that the complexity of a method, and therefore the effort of applying it, increases with the number of nodes, this indicates that most of the classified alignment methods require a relatively higher effort in the start-up phase (RE) than in follow-up phases.

While the REST taxonomy provides a framework for classifying alignment methods, the idea that dyad structures exhibit advantages but also liabilities with respect to the coordination between RE and ST led to the development of REST-bench, a lightweight assessment method. REST-bench is an interview-driven, cooperative assessment method that aims at identifying gaps and bottlenecks in REST alignment, as observed in software development projects. We first proposed and piloted REST-bench in the study presented in Chapter 3, while validating and improving the method in Chapter 4. As such, the validation of REST-bench, in terms of the ability in identifying REST alignment improvement potential, provides also evidence for the validity of the constructs that define the REST taxonomy.

With respect to the overall research goal RG-1, understand REST alignment, the concepts elaborated in Chapter 3 form the basis for the remainder of this thesis. In fact, the information dyad and structures form the conceptual framework [Wieringa, 2014a] for the subsequent research, either explicitly (Chapter 4) or implicitly (Chapters 5, 6 and 7).

**MAIN FINDINGS.** The central characteristic of a REST alignment method is which and how information is linked such that both RE and ST activities can be coordinated. Information dyads, which formalize this characteristic,

help to both understand and analyze REST alignment methods, and serve as well as analytic tool for assessing the state of REST alignment.

### 5.3 *Chapter 4*

The premise of this chapter is that for any improvement, it is necessary to characterize the current condition of the object under study [Ares et al., 2000]. Based on this agreed state and the definition of goals, changes can be designed and implemented. In the past, postmortems [Birk et al., 2002] have been proposed to identify best practices and issues in the execution of projects. These can then be fed into an organizational knowledge repository [Ivarsson and Gorscak, 2012]. While guidelines for executing postmortems exist, e.g by Collier et al. [1996] and Dingsøyr [2005], post-mortem reviews are seldom held even though their benefits are well reported [Verner and Evanco, 2005]. Some suggest that lack of time is a major detractor to perform project postmortems [Keegan and Turner, 2001, Glass, 2002a].

Therefore, we designed REST-bench to be lightweight, in terms of resource use, by focusing the assessment effort to the specific issue of REST alignment. In Chapter 4, we present REST-bench in an example-driven manner, describing data elicitation, data preparation and the collaborative analysis that involves all assessment participants. The main idea of REST-bench is to analyze artifact creation and use, from the perspective of requirements and test engineers, in order to determine how information is shared, which in turn is central to coordination [van de Ven et al., 1976] and can be analyzed by means of dyad structure properties. We acknowledge that artifacts are only one way to coordinate; Stapel et al. [2011], for example, differentiate between solid (long term and repeatably accessible) and fluid (volatile, implicit) information containers. The analysis of fluid information, shared through informal emails, instant messages, telephone calls and meetings is however not the focus of REST-bench. Therefore, we illustrate the application of REST-bench in five companies that exhibit diverse characteristics (with complex, documentation driven projects, but also agile instances) and identify concrete improvement opportunities. The participants of the assessment, both from the plan-driven and the Agile spectrum, judged REST-bench as an efficient and effective mean to assess the coordination between RE and ST.

We conducted this research to provide support for the case companies, but also to verify the conceptual framework, i. e. the constructs of the REST taxonomy, developed in Chapter 3. We did that by identifying those dyad structure properties that were helpful in defining seeding questions for the REST-bench analysis process. From the properties defined in Table 1.3, all but P<sub>3</sub> (intermediate nodes) and P<sub>4</sub> (RE and ST node proportion) led to

useful seeding questions and analysis points, contributing to the identification of improvement opportunities at the assessed companies.

**MAIN FINDINGS.** REST-bench, based on eliciting and analyzing artifact creation and use, identified REST alignment improvement opportunities both in plan-driven and as well as Agile projects. Practitioners judged the method as efficient and effective. Furthermore, four out of six dyad structure properties were found to be useful to generate assessment questions, providing empirical evidence for the validity of the REST taxonomy on which this assessment method is based on.

#### 5.4 *Chapter 5*

The premise of the study reported in this chapter is that extensive requirements documentation can be substituted, under certain circumstances, by test cases. The information provided in test cases, in particular on the system and acceptance test level, reflects to a large degree the intention and purpose of the system under test. Therefore, while requirements documents would be still useful during early stages of a development project, i. e. in elicitation and analysis, they could be replaced by test cases which provide value during and after development, i. e. in maintenance. We have observed this practice, using test cases as requirements (TCR), in the case study reported in Chapter 2 and analyze in this chapter motivations, advantages, but also liabilities of applying this practice, from the perspective of practitioners. Note that, while the companies in which we observed the practice is a subset of the companies studied in Chapter 2, we re-analyzed the data to answer more narrow research questions (compare RQ-1.2 to RQ-4.1 and RQ-4.2 in Figure 1.2).

**MAIN FINDINGS.** We identified three scenarios in which we observed the TCR practice: (1) as a de facto standard, (2) as an established practice and (3) as a planned practice as part of an Agile transition. In all three scenarios, the practice provides benefits but induces also major challenges. One common challenge encountered in all three scenarios is that a unification of artifacts does not necessarily lower communication barriers raised by the fact that stakeholders have varying information needs. In other words, when applying the TCR practice, for whatever particular reason, there is also an increased need for coordination through informal means (ad-hoc meetings, email, phone calls), favored by an Agile, co-located working environment.

## 5.5 *Chapter 6*

In this and the following chapter we focus on developing and validating a solution that addresses some REST alignment challenges identified in Chapters 2 and 5. Since this research was conducted at one particular case company (see Figure 1.5), the developed solutions are connected mostly to Company A that applies the practice of using test cases as requirements, leading to challenges in requirements change communication and test case selection (see Chapter 5).

The premise of the studies reported both in this and the next chapter is that we can support engineers in test case selection by employing information retrieval techniques that recover traceability links between existing, but originally not connected artifacts. These traceability links would then allow engineers, depending on the situation, allow better test case selection, either by choosing only relevant test cases (focus of the approach in Chapter 6) or identifying unexpected or forgotten test cases (the focus on the approach in Chapter 7).

The main goal of the study reported in Chapter 6 is to illustrate the path from problem identification to solution definition and validation, and to dissect the taken design decisions that address a real-world software engineering problem. The solution approach we evaluate in a series of three experiments is based on the idea that we can recover traceability links between source code, in particular product variants where certain features can be activated and deactivated, and test cases that verify those features. We evaluate several information retrieval (IR) technique configurations, varying the input data, preprocessing steps and IR models. Each of the three experimental setups led to lessons learned that steered the design of the next experiment. The main challenge we encountered in implementing IR techniques on large industry-scale datasets was exhaustive parameter optimization in order to find the optimal technique configuration. Looking at the decision support the approach can potentially provide, we found that the results are comparable, in terms of mean average precision, with previous research aiming at automating trace recovery.

**MAIN FINDINGS.** Automated trace recovery, for the described use case in an industry setting, is challenging for several reasons. First and foremost, one has to consider the trade-off between using a realistic dataset and exhaustive parameter optimization. In other words, with current technology, one can either choose to use industry-grade datasets and evaluate only a subset of IR technique configurations (for Vector Space Models [Salton et al., 1975] and Latent Semantic Analysis models [Deerwester et al., 1990]), or reduce the dataset and perform exhaustive parameter tuning. This is however a problem since our results indicate that, with the size and dif-

ficulty of the trace recovery task, performance of IR configurations varies, i.e. is not stable with respect to the given input data.

## 5.6 *Chapter 7*

The findings from Chapter 6 led us to reconsider the overall approach for test case selection decision support based on source code changes induced by feature activation and deactivation. We tried to identify the reasons why the approach seems to work better for some features than for others. In the course of this investigation, we also analyzed how similar system test cases are to each other. We observed that, by using a simple topic modeling [Blei et al., 2003] approach, that the clusters of similar test cases mirrored approximately the test modules in which they were originally categorized by test engineers. However, we observed also that test cases belonging to different test modules were clustered together. This sparked the idea for the study reported in Chapter 7 whose premise is that we can exploit the properties of topic models to support test engineers in selecting test cases they did not initially consider or simply forgot.

Risk-based testing supported by topic models (RiTTM) is a system test case selection approach that provides decision support based on existing artifacts and resources that are readily available. RiTTM does not introduce new documentation or models that require manual maintenance. The model selection is automated such that, depending on the input data, the best topic model, in terms of interpretability, can be generated. While model selection is automated, the test case selection process itself integrates the engineer; hence, RiTTM is designed to enhance human expertise rather than trying to replace or mimicry it.

**MAIN FINDINGS.** A non-domain expert identified 29 relevant test cases that were previously not selected by a test expert. The majority of these test cases (59%) were found to be categorized in test modules that the test engineer regarded irrelevant for the particular features that were to be tested. This preliminary evaluation provides evidence that topic models, and RiTTM in particular, can support test engineers in identifying overlooked or forgotten test cases.

## 6 SYNTHESIS

We return now to the initially stated research goals (Section 3) and discuss the level to which they have been achieved, both in the context of the contributions in this thesis and as well as in related work. Throughout

this section, we use also the information dyad concept, summarized in Section 5.2, as an analytical tool<sup>6</sup>.

### 6.1 RG-I: Understand REST Alignment

The recognition of the importance of REST alignment is *not* a contribution of this thesis. Research preceding this work has identified this area as an important object of scientific study [Cheng and Atlee, 2007, Bertolino, 2007] but also as relevant for practitioners [Graham, 2002, Uusitalo et al., 2008, Martin and Melnik, 2008, Post et al., 2009, Kukkanen et al., 2009]. The study of REST alignment challenges and practices (Chapter 2) indicates that practitioners are well aware of the consequences of (mis-)alignment. This allowed us to better understand the context of REST (mis-)alignment, an important factor for developing improvement solutions. However, the underlying mechanisms that cause misalignment or support alignment were not identified in this case study. The information dyad and dyad structures concepts, introduced in Chapter 3, provide a mean to describe how and why REST alignment methods work or provide an explanation why a lack of alignment exists in an observed situation. The REST taxonomy is one (incomplete) candidate theory that furthers our understanding of alignment, another is the distance theory developed later by Bjarnason et al. [2014b]. The distance theory also considers coordination through information as a central mechanism in alignment, focuses however on characterizing the distances between entities (which could be described as nodes in the context of the REST taxonomy). Distances can be geographical, temporal, organizational, psychological, cognitive, adherent, semantic, and navigational [Bjarnason et al., 2014b]. However, an aspect that still needs to be defined are the measurement units for these distances. For example, semantic distance could be assessed by similarity measures based on term frequencies (e.g. the Vector Space Model [Salton et al., 1975], as used in Chapter 6). Other distances, such as psychological or cognitive distance, would require additional research in order to understand how they could be measured in the context of software engineering. Once defined with measures, the distance concept could provide an additional, orthogonal dimension describing the link between nodes in an information dyad.

We continue our discussion on how the REST taxonomy furthers our understanding of alignment by analyzing the evolution of dyad structures encountered at a specific company. As Table 1.2 illustrates, Company A participated in the studies reported in Chapter 2 and 5, where their REST alignment challenges and practices are illustrated, and in Chapters 6 and 7, where we developed solutions to address these challenges.

---

<sup>6</sup> For a full appreciation of this section, we recommend to read it again after having read the remaining chapters of this thesis.

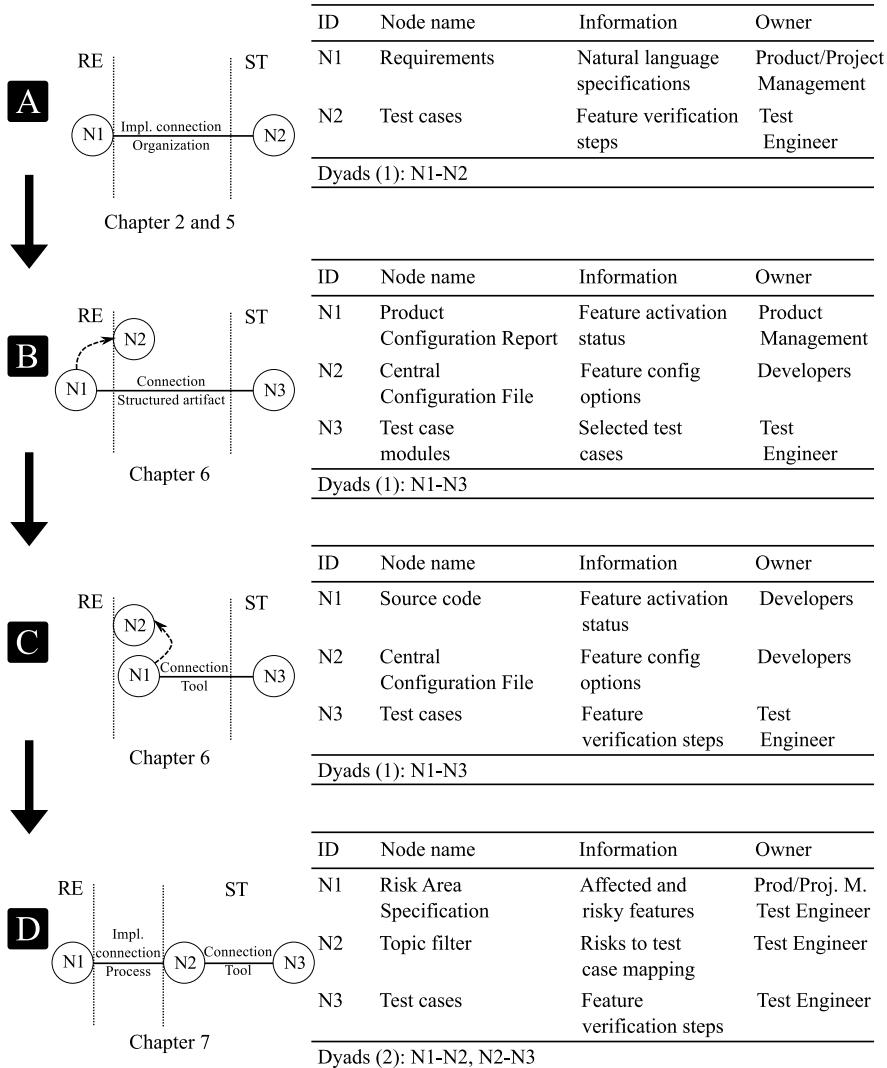


Figure 1.7: Evolution of dyad structures encountered and developed in solutions at Company A

Panel A in Figure 1.7 illustrates our interpretation, in terms of the REST taxonomy constructs, of one challenge encountered at Company A. The link between the two nodes (N<sub>1</sub>–Requirements, N<sub>2</sub>–Test cases) is established by an implicit connection, that is, “having a line of thought (not necessarily documented)” (Section 4.1.7 in Chapter 2). The benefits of the approach shown in Panel A is lightweight requirements management which makes, at the same time, the estimation of test case coverage difficult (see Section 5.1 in Chapter 5), leading to reduced confidence in quality assurance (see Section 2.1.1 in Chapter 6). Traceability provides confidence [Panis, 2010], is however costly to establish and, more importantly, to maintain [Heindl and Biffl, 2005]; traceability needs to have a clear purpose and benefit [Dömges and Pohl, 1998]. When traceability is not planned for in advance, it is possible to recover traces from the existing information [Gotel et al., 2012]. Panels B–C in Figure 1.7, discussed next, represent such traceability recovery approaches observed and developed at Company A.

As described in Section 2.1.2 in Chapter 6, Company A developed an approach to establish traceability, illustrated in Panel B in Figure 1.7. Compared to the previous approach, shown in Panel A, the solution in Panel B establishes an explicit link between features (N<sub>1</sub>–Product Configuration Report (PCR)) and selected test cases (N<sub>3</sub>–Test case modules). On the other hand, the granularity in the PCR allows only a mapping to test modules which can contain a large set of test cases. Furthermore, the mapping between PCR and test modules needs to be maintained manually by product experts (see Section 2.1.2 in Chapter 6).

These drawbacks led to development of the solution shown in Panel C in Figure 1.7. In Chapter 6 we report on three experimental iterations where we studied the trace recovery between source code (N<sub>1</sub>–Feature activation status) and test cases (N<sub>3</sub>–Selected test cases). Compared to the solution shown in Panel B, this approach is fully automatic and granular to the test case level. However, low accuracy and computational inefficiency renders this approach less feasible in practice (see Section 4.3.5 in Chapter 6). Alternative approaches suggest that one could add information sources, i. e. by combining textual with runtime [Qusef et al., 2014] or textual with structural information [McMillan et al., 2009], or combine recovery methods (e.g. Gethers et al. [2011], Falessi et al. [2013]), improving accuracy of trace recovery by triangulation.

The insights gained from the approach evaluated in Chapter 6 led to the development of the solution shown in Panel D in Figure 1.7. Here, we support the implicit connection between risk areas (N<sub>1</sub>–Affected and risky product features), a generated topic filter (N<sub>2</sub>–Risks to test case mapping), and the connection to test cases (N<sub>3</sub>–Feature verification steps). In this approach, we added a node (N<sub>2</sub>) to the company’s artifacts that facilitates the trace recovery, rather than trying to automate it (as in the approach shown in Panel C). Note that, while the connection between N<sub>1</sub> and N<sub>2</sub>

is implicit, i. e. relies upon the knowledge of the engineer who applies the approach, the process of establishing the connection is explicit and can, as shown in Chapter 7, be performed by a non-expert. This is in contrast to the situation shown in Panel A, where we also observed an implicit connection, established by an organizational structure, role responsibilities and individual expert knowledge on the domain.

In conclusion, we can state that the REST taxonomy, in particular the information dyad construct, provides a useful mean to characterize alignment approaches (or lack thereof) and aspects, such as information use, creation and linking, that are central for alignment. Next, we discuss the utility of the taxonomy for the second research goal.

## 6.2 RG-II: Improve REST Alignment

Looking at the assessments performed with REST-bench, illustrated in Chapter 4, one commonality encountered in all five cases is the identification of improvement possibilities in the linking mechanisms between nodes of information. This means either to establish a link in the first place, or improve an existing link such that it is more resilient, accurate or reliable over time. In some cases, a solution would be to introduce explicit traces between artifacts, either by adopting a viable trace model [Ramesh and Jarke, 2001] or by automatically recovering trace links (see Borg et al. [2014] for an overview of techniques). In other cases, the involvement of test engineers in requirements engineering tasks would already provide considerable benefits, as observed by Damian and Chisan [2006], Uusitalo et al. [2008] and Kukkanen et al. [2009]. Generally, the assessed companies were well aware that information either from RE or ST is not used to its full potential for decision support. Consequentially, the study participants valued REST-bench's ability to identify gaps in the coordination between RE and ST. Similar observations, focused on gaps between RE and downstream development affecting test coverage and scoping were made by Bjarnason et al. [2011] in the context of large-scale software development. We have applied REST-bench also in Agile [Beck et al., 2001, Dybå and Dingsøyr, 2008] contexts, and the identification of gaps in the coordination between RE and ST was considered important by the study participants. At a first glance, this might be surprising, since Agile approaches tend to favor coordination via co-location and frequent face-to-face communication [Begel and Nagappan, 2007, Mishra et al., 2012] rather than via artifacts, as assessed with REST-bench. However, Strode et al. [2012] have studied how Agile teams coordinate, suggesting that they employ a synchronization strategy through activities that produce artifacts containing information used by all team members (a theory also supported by the work on the role of physical artifacts in Agile development [Sharp et al., 2009]). Fur-

thermore, work by [Pikkarainen et al. \[2008\]](#) suggests that Agile teams use also plan-driven practices to communicate efficiently with external actors of software development. These observations provide a likely explanation why the application of REST-bench, based on studying use and creation of artifacts, led to useful results both in plan-driven and as well in Agile environments.

We turn now the discussion to the studies presented in Chapters [6](#) and [7](#) which also aimed at improving REST alignment by changing the linking mechanism between information nodes. Panel A and B in Figure [1.7](#) illustrate the starting situation while panels C and D show the treatments we implemented and evaluated at Company A (we discussed the solution evolution in Section [6.1](#)). In both Chapter [6](#) and [7](#), we aimed to provide decision support for test case selection to test engineers. In Chapter [6](#) we developed a white-box approach, i.e. we used the product itself (source code of product variants and product configuration files) to recover traceability to test cases. This approach assumes that source code (identifier names, literals and comments) and test cases share a similar, domain-specific, vocabulary. This is a basic assumption where trace recovery between artifacts at different abstraction levels is attempted [[Antoniol et al., 2002](#)]. The final iteration of the three experiments in Chapter [6](#) produced a ranking of test cases relevant for a particular product variant. The average precision of suggestions, while comparable to similar studies [[Abadi et al., 2008](#), [Cleary et al., 2009](#), [Qusef et al., 2014](#), [Xia et al., 2015](#)], was however too low to provide test engineers confidence in the ranking and value using the method. In hindsight, we can spot two issues with the approach evaluated in Chapter [6](#). First of all, the original problem we were trying to address was the selection of test cases that are applicable to particular product variants; this task required high precision (high relevance in the top N ranked test cases) in order provide added value to expert test engineers, which we could not deliver with the evaluated approach. A second hurdle to the practical application was the computational inefficiency we encountered in analyzing large source code corpora. This prevented an exhaustive evaluation of IR technique configurations.

At this point, we can ask: did we try to solve the right problem with the appropriate means? We may shed some light on this by analyzing the solutions by means of information dyads. The Product Configuration Report ( $N_1$ , see Panel B in Figure [1.7](#), representing the solution developed at Company A that we attempted to improve in Chapter [6](#)), is an artifact used by test engineers but also by product managers. Both engineers and managers review the report, forming an understanding on the product features and how they should or can be tested. The solution illustrated in Panel C in Figure [1.7](#) attempted to recover traces between source code ( $N_1$ ) and test cases ( $N_3$ ), replacing the need to manually map a feature to test cases (as in the solution in Panel B). However, this removed product management as a

direct actor, and with that an important resource of information (see Panel C). In hindsight, and from the perspective of the information dyads analysis, the solution developed in Chapter 6 would require, besides addressing the technical deficiencies discussed earlier, also some design refinements (involve product and/or project management) in order to actually improve REST alignment.

Using the same means, we analyze now the solution illustrated in Panel D in Figure 1.7 and presented in Chapter 7. Here we recover traces between test cases ( $N_3$ ) and an artifact used by both managers and engineers, the risk area specification ( $N_1$ ). In principle, recovering traces between these two artifacts, connecting RE and ST, should have a larger potential in improving REST alignment than the approach shown in Panel C, where no RE artifacts or roles are involved. The goal of recovering traces between risk areas and test cases was to improve test coverage. Note that this subtle change in problem framing changed also the requirements for the solution. While in Chapter 6 high precision was required, the solution in Chapter 7 needed high recall in order to be of value for test engineers.

## 7 CONCLUSIONS AND FUTURE WORK

This thesis aimed at contributing to the understanding of coordinating requirements engineering (RE) and software testing (ST) activities and as well as providing means for their improvement in industry. Work towards the first goal included: (1) a large observational case study where we identified REST alignment challenges and practices, and (2) the creation of a taxonomy of REST alignment methods that led also to the definition of the information dyad. Work towards the second goal included: (3) a lightweight assessment framework, REST-bench, and (4) an investigation into the potential of information retrieval techniques to recover traces for the purpose of supporting test case selection.

With respect to the first goal, understanding REST alignment, we conclude that the identified alignment challenges are as diverse as the practices that potentially address them. It is therefore convenient to develop a conceptual framework that helps to study alignment methods at a common level. The information dyad and the resulting dyad structure properties turned out to be useful in this regard. A potential threat of creating such abstractions is that relevant dimensions are not represented, leading to a misunderstanding of the real-life phenomenon under study. We addressed this threat by operationalizing the conceptual framework in an assessment method that confirmed the underlying constructs by supporting the identification of relevant improvement opportunities in REST alignment. While we have developed the basis, in the form of constructs and a defined scope [Sjøberg et al., 2008], for a theory, there are still some components missing

that are required for the formulation of an REST alignment theory. For example, propositions define how constructs interact, allowing to formulate testable hypotheses [Sjøberg et al., 2008]. The dyad structure properties we identified can serve to formulate testable propositions. Some properties like P<sub>1</sub>, P<sub>2</sub> and P<sub>4</sub> (see Table 1.3) are already formulated as propositions and could be tested by further and continued applications of REST-bench. The second missing component are explanations for these propositions, which would allow engineers to design alignment methods. By defining these missing components, the REST alignment theory could be compared and/or combined with other related theories, e.g. with the theory on co-ordination of engineering decisions [Herbsleb and Mockus, 2003] or the theory on distances affecting alignment practices [Bjarnason et al., 2014b].

With respect to the second goal, improve REST alignment, we conclude that we covered some ground by formulating, implementing and validating a framework for assessing REST alignment. In the design of REST-bench, we focused on efficiency for the applying organization. We regard this, besides effectiveness, as an important factor for the adoption by industry. Our immediate future work targets the bundling of REST-bench with instructions and tool support such that it can be used “as-is” by industry. This would provide additional empirical data needed to test some of the dyad structure properties.

We evaluated the idea of improving REST alignment by recovering trace links to support test case selection, based on two different approaches: (1) source code differences between product variants, and (2) risk areas descriptions. The first approach attempted to replace a manual tracing procedure, requiring high precision which was not achieved with the available techniques. Future work in this direction includes the consideration of more information sources, e.g. the version control commit history, which could increase the precision of the trace recovery process. Furthermore, it would be interesting to evaluate whether we can identify optimal information retrieval technique configurations by applying the approach proposed by Lohar et al. [2013]. The second approach attempted to support test engineers in their selection of test cases, that is, augment their ability to find and select relevant test cases for a particular high risk feature in the product. The preliminary evaluation of this idea showed promising results as it enabled a non-domain expert to identify relevant test cases that a test expert did initially not choose. However, further work is needed to estimate the cost-benefit ratio of increasing the test coverage with this approach.

# 2

---

## CHALLENGES AND PRACTICES IN ALIGNING REQUIREMENTS WITH VERIFICATION AND VALIDATION: A CASE STUDY OF SIX COMPANIES

---

**SUMMARY** Weak alignment of requirements engineering (RE) with software testing (ST)<sup>1</sup> may lead to problems in delivering the required products in time with the right quality. For example, weak communication of requirements changes to testers may result in lack of verification of new requirements and incorrect verification of old invalid requirements, leading to software quality problems, wasted effort and delays. However, despite the serious implications of weak alignment research and practice both tend to focus on one or the other of RE or ST rather than on the alignment of the two. We have performed a multi-unit case study to gain insight into issues around aligning RE and ST by interviewing 30 practitioners from 6 software developing companies, involving 10 researchers in a flexible research process for case studies. The results describe current industry challenges and practices in aligning RE with ST, ranging from quality of the individual RE and ST activities, through tracing and tools, to change control and sharing a common understanding at strategy, goal and design level. The study identified that human aspects are central, i. e. cooperation and communication, and that requirements engineering practices are a critical basis for alignment. Further, the size of an organization and its motivation for applying alignment practices, e. g. external enforcement of traceability, are variation factors that play a key role in achieving alignment. Our results provide a strategic road-map for practitioners improvement work to address alignment challenges. Furthermore, the study provides a foundation for continued research to improve the alignment of RE with ST.

### 1 INTRODUCTION

Requirements engineering (RE) and software testing (ST) both aim to support development of products that will meet customers' expectations re-

---

<sup>1</sup> Note that, in order to maintain consistency with the remaining chapters, we changed the terms "verification and validation" from the original paper with "software testing" throughout this chapter.

garding functionality and quality. However, to achieve this RE and ST need to be aligned and their “activities or systems organized so that they match or fit well together” (MacMillan Dictionary’s definition of “align”). When aligned within a project or an organization, RE and ST work together like two bookends that support a row of books by buttressing them from either end. RE and ST, when aligned, can effectively support the development activities between the initial definition of requirements and acceptance testing of the final product [Damian and Chisan, 2006].

Weak coordination of requirements with development and testing tasks can lead to inefficient development, delays and problems with the functionality and the quality of the produced software, especially for large-scale development [Kraut and Streeter, 1995]. For example, if requirements changes are agreed without involving testers and without updating the requirements specification, the changed functionality is either not verified or incorrectly verified. This weak alignment of RE and work that is divided and distributed among engineers within a company or project poses a risk of producing a product that does not satisfy business and/or client expectations [Gorschek and Davis, 2008]. In particular, weak alignment between RE and ST may lead to a number of problems that affect the later project phases such as non-verifiable requirements, lower product quality, additional cost and effort required for removing defects [Sabalaiuskaite et al., 2010]. Furthermore, Jones et al. [2009] identified three other alignment related problems found to affect independent testing teams, namely uncertain test coverage, not knowing whether changed software behavior is intended, and lack of established communication channels to deal with issues and questions.

There is a large body of knowledge for the separate areas of RE and ST, some of which touches on the connection to the other field. However, few studies have focused specifically on the alignment between the two areas [Barmi et al., 2011] though there are some exceptions. Kukkanen et al. [2009] reported on lessons learnt in concurrently improving the requirements and the testing processes based on a case study. Another related study was performed by Uusitalo et al. [2008] who identified a set of practices used in industry for linking requirements and testing. Furthermore, RE alignment in the context of outsourced development has been pointed out as a focus area for future RE research by Cheng and Atlee [2007].

When considering alignment, traceability has often been a focal point [Watkins and Neal, 1994, Barmi et al., 2011, Paci et al., 2012]. However, REST alignment also covers the coordination between roles and activities of RE and ST. Traceability mainly focuses on the structuring and organization of different related artifacts. Connecting (or tracing) requirements with the test cases that verify them support engineers in ensuring requirements coverage, performing impact analysis for requirements changes etc. In addition to tracing, alignment also covers the interaction between roles

throughout different project phases; from agreeing on high-level business and testing strategies to defining and deploying detailed requirements and test cases.

Our case study investigates the challenges of RE and ST (REST) alignment, and identifies methods and practices used, or suggested for use, by industry to address these issues. The results reported in this chapter are based on semi-structured interviews of 90 minutes each with 30 practitioners from six different software companies, comprising a wide range of people with experience from different roles relating to RE and ST. This chapter extends on preliminary results of identifying the challenges faced by one of the companies included in our study [Sabaliauskaite et al., 2010]. In this chapter, we report on the practices and challenges of all the included companies based on a full analysis of all the interview data. In addition, the results are herein categorized to support practitioners in defining a strategy for identifying suitable practices for addressing challenges experienced in their own organizations.

The rest of this chapter is organized as follows: Section 2 presents related work. The design of the case study is described in Section 3, while the results can be found in Section 4. In Section 5 the results are discussed and, finally the chapter is concluded in Section 6.

## 2 RELATED WORK

The software engineering fields RE and ST have mainly been explored with a focus on one or the other of the two fields [Barmi et al., 2011], though there are some studies investigating the alignment between the two. Through a systematic mapping study into alignment of requirements specification and testing, Barmi et al. found that most studies in the area were on model-based testing including a range of variants of formal methods for describing requirements with models or languages from which test case are then generated. Barmi et al. also identified traceability and empirical studies into alignment challenges and practices as main areas of research. Only three empirical studies into REST alignment were found. Of these, two originate from the same research group and the third one is the initial results of the study reported in this chapter. Barmi et al. draw the conclusions that though the areas of model-based engineering and traceability are well understood, practical solutions including evaluations of the research are needed. In the following sections previous work in the field is described and related to this study at a high level. Our findings in relation to previous work are discussed in more depth in Section 5.

*The impact of RE on the software development process* as a whole (including testing) has been studied by Damian et al. [2005] who found that improved RE and involving more roles in the RE activities had positive effects

on testing. In particular, the improved change control process was found to “bring together not only the functional organization through horizontal alignment (designers, developers, testers and documenters), but also vertical alignment of organizational responsibility (engineers, teams leads, technical managers and executive management)” [Damian et al., 2005]. Furthermore, in another study [Damian and Chisan, 2006] found that rich interactions between RE and testing can lead to pay-offs in improved test coverage and risk management, and in reduced requirements creep, over-scoping and waste, resulting in increased productivity and product quality. Gorscheck and Davis [2008] have proposed a taxonomy for assessing the impact of RE on, not just project, but also on product, company and society level; to judge RE not just by the quality of the system requirements specification, but also by its wider impact.

*Jointly improving the RE and testing processes* was investigated by Kukkanen et al. [2009] through a case study on development performed partly in the safety-critical domain with the dual aim of improving customer satisfaction and product quality. They report that integrating requirements and testing processes, including clearly defining RE and testing roles for the integrated process, improves alignment by connecting processes and people from requirements and testing, as well as, applying good practices that support this connection. Furthermore, they report that the most important aspect in achieving alignment is to ensure that “the right information is communicated to the right persons” [Kukkanen et al., 2009]. Successful collaboration between requirements and test can be ensured by assigning and connecting roles from both requirements and test as responsible for ensuring that reviews are conducted. Among the practices implemented to support requirements and test alignment were the use of metrics, traceability with tool support, change management process and reviews of requirements, test cases and traces between them [Kukkanen et al., 2009]. The risk of overlapping roles and activities between requirements and test, and gaps in the processes was found to be reduced by concurrently improving both processes [Kukkanen et al., 2009]. These findings correlate very well with the practices identified through our study.

*Alignment practices* that improve the link between requirements and test are reported by Uusitalo et al. [2008] based on six interviews, mainly with test roles, from the same number of companies. Their results include a number of practices that increase the communication and interaction between requirements and testing roles, namely early tester participation, traceability policies, consider feature requests from testers, and linking test and requirements people. In addition, four of the companies applied traceability between requirements and test cases, while admitting that traces were rarely maintained and were thus incomplete [Uusitalo et al., 2008]. Linking people or artifacts were seen as equally important by the interviewees who were unwilling to select one over the other. Most of the practices

reported by Uusitalo et al. were also identified in our study except for the specific practice of linking testers to requirements owners and the practice of including internal testing requirements in the project scope.

*The concept of traceability* has been discussed, and researched since the very beginning of software engineering, i. e. since the 1960s [Randell, 1968]. Traceability between requirements and other development artifacts can support impact analysis [Gotel and Finkelstein, 1994, Watkins and Neal, 1994, Ramesh et al., 1997, Damian et al., 2005, Uusitalo et al., 2008, Kukkanen et al., 2009], lower testing and maintenance costs [Watkins and Neal, 1994, Kukkanen et al., 2009], and increased test coverage [Watkins and Neal, 1994, Uusitalo et al., 2008] and thereby quality in the final products [Watkins and Neal, 1994, Ramesh et al., 1997]. Tracing is also important to software verification due to being an (acknowledged) important aspect in high quality development [Watkins and Neal, 1994, Ramesh et al., 1997]. The challenges connected to traceability have been empirically investigated and reported over the years. The found challenges include volatility of the traced artifacts, informal processes with lack of clear responsibilities for tracing, communication gaps, insufficient time and resources for maintaining traces in combination with the practice being seen as non-cost efficient, and a lack of training [Cleland-Huang et al., 2003]. Several methods for supporting automatic or semi-automatic recovery of traces have been proposed as a way to address the cost of establishing and maintaining traces, e.g. De Lucia et al. [2007], Hayes et al. [2007], Lormans et al. [2008]. An alternative approach is proposed by Post et al. [2009] where the number of traces between requirements and test are reduced by linking test cases to user scenarios abstracted from the formal requirements, thus tracing at a higher abstraction level. When evaluating this approach, errors were found both in the formal requirements and in the developed product [Post et al., 2009]. However, though the evaluation was performed in an industrial setting the set of 50 requirements was very small. In conclusion, traceability in full-scale industrial projects remains an elusive and costly practice to realize [Gotel and Finkelstein, 1994, Watkins and Neal, 1994, Jarke, 1998, Ramesh, 1998]. It is interesting to note that Gotel and Finkelstein conclude that a “particular concern” in improving requirements traceability is the need to facilitate informal communication with those responsible for specifying and detailing requirements. Another evaluation of the traceability challenge reported by Ramesh [1998] identifies three factors as influencing the implementation of requirements traceability, namely environmental (tools), organizational (external organizational incentive on individual or internal), and development context (process and practices).

*Model-based testing* is a large research field within which a wide range of formal models and languages for representing requirements have been suggested [Dias Neto et al., 2007]. Defining or modeling the requirements in a formal model or language enables the automatic generation of other

development artifacts such as test cases, based on the (modeled) requirements. Similarly to the field of traceability, model-based testing also has issues with practical applicability in industrial development [Nebut et al., 2006, Mohagheghi and Dehlen, 2008, Yue et al., 2011]. Two exceptions to this is provided by Hasling et al. [2008] and by Nebut et al. [2006] who both report on experiences from applying model-based testing by generating system test cases from UML descriptions of the requirements. The main benefits of model-based testing are in increased test coverage [Nebut et al., 2006, Hasling et al., 2008], enforcing a clear and unambiguous definition of the requirements [Hasling et al., 2008] and increased testing productivity [Grieskamp et al., 2011]. However, the formal representation of requirements often results in difficulties both in requiring special competence to produce [Nebut et al., 2006], but also for non-specialist (e.g. business people) in understanding the requirements [Lubars et al., 1993]. Transformation of textual requirements into formal models could alleviate some of these issues. However, additional research is required before a practical solution is available for supporting such transformations [Yue et al., 2011]. The generation of test cases directly from the requirements implicitly links the two without any need for manually creating (or maintaining) traces. However, depending on the level of the model and the generated test cases the value of the traces might vary. For example, for use cases and system test cases the tracing was reported as being more natural than when using state machines [Hasling et al., 2008]. Errors in the models are an additional issue to consider when applying model-based testing [Hasling et al., 2008]. Scenario-based models where test cases are defined to cover requirements defined as use cases, user stories or user scenarios have been proposed as an alternative to the formal models, e.g. by Regnell and Runeson [1998], Regnell et al. [2000] and Melnik et al. [2006]. The scenarios define the requirements at a high level while the details are defined as test cases; acceptance test cases are used to document the detailed requirements. This is an approach often applied in agile development [Cao and Ramesh, 2008]. Melnik et al. [2006] found that using executable acceptance test cases as detailed requirements is straight-forward to implement and breeds a testing mentality. Similar positive experiences with defining requirements as scenarios and acceptance test cases are reported from industry by Martin and Melnik [2008].

### 3 CASE STUDY DESIGN

The main goal of this case study was to gain a deeper understanding of the issues in REST alignment and to identify common practices used in industry to address the challenges within the area. To this end, a flexible exploratory case study design [Robson, 2002, Runeson et al., 2012] was

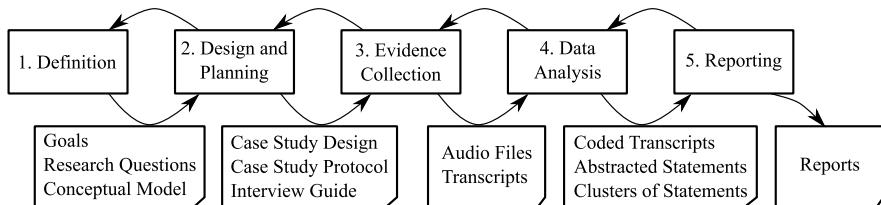


Figure 2.1: Overview of the research process including in- and output for each phase

chosen with semi-structured interviews as the data collection method. In order to manage the size of the study, we followed a case study process suggested by Runeson et al. [2012, chapter 14] which allowed for a structured approach in managing the large amounts of qualitative data consistently among the many researchers involved. The process consists of the following five interrelated phases (see Figure 2.1 for an overview, including in- and outputs of the different phases):

1. *Definition* of goals and research questions
2. *Design and planning* including preparations for interviews
3. *Evidence collection* (performing the interviews)
4. *Data analysis* (transcription, coding, abstraction and grouping, interpretation)
5. *Reporting*

Phases 1–4 are presented in more detail in Sections 3.1 to 3.4, while threats to validity are discussed in Section 3.5. A more in-depth description with lessons learned from applying the process in this study is presented by Runeson et al. [2012, chapter 14]. A description of the six case companies involved in the study can be found in Section 3.2.

The ten authors played different roles in the five phases shown in Figure 2.1. The senior researchers, Regnell, Gorschek, Runeson and Feldt led the *goal definition* of the study. They also coached the *design and planning*, which was practically managed by Loconsole, Sabaliauskaite and Engström. *Evidence collection* was distributed over all ten researchers. Loconsole and Sabaliauskaite did the transcription and coding together with Bjarnason, Borg, Engström and Unterkalmsteiner, as well as the preliminary *data analysis* for the evidence from the first company (Sabaliauskaite et al. [2010]). Bjarnason, Borg, Engström and Unterkalmsteiner did the major legwork in the intermediate data analysis, coached by Regnell, Gorschek and Runeson. Bjarnason and Runeson made the final *data analysis, interpretation* and *reporting*, which was then reviewed by the rest of the authors.

### *3.1 Definition of Research Goal and Questions*

This initial phase (see Figure 2.1) provided the direction and scope for the rest of the case study. A set of goals and research questions were defined based on previous experience, results and knowledge of the participating researchers, and a literature study into the area. The study was performed as part of an industrial excellence research centre, where REST alignment was one theme. Brainstorming sessions were also held with representatives from companies interested in participating in the study. In these meetings the researchers and the company representatives agreed on a main long-term research goal for the area: to improve development efficiency within existing levels of software quality through REST alignment, where this case study takes a first step into exploring the current state of the art in industry.

Furthermore, a number of aspects to be considered were agreed upon, namely agile processes, open source development, software product line engineering, non-functional requirements, and, volume and volatility of requirements. As the study progressed the goals and focal aspects were refined and research questions formulated and documented by two researchers. Four other researchers reviewed their output. Additional research questions were added after performing two pilot interviews (in the next phase, see Section 3.2). In this chapter, the following research questions are addressed in the context of software development:

**RQ1:** What are the current challenges, or issues, in achieving REST alignment?

**RQ2:** What are the current practices that support achieving REST alignment?

**RQ3:** Which current challenges are addressed by which current practices?

The main concepts of REST alignment to be used in this study were identified after discussions and a conceptual model of the scope of the study was defined (see Figure 2.2). This model was based on a traditional V-model showing the artifacts and processes covered by the study, including the relationships between artifacts of varying abstraction level and between processes and artifacts. The discussions undertaken in defining this conceptual model led to a shared understanding within the group of researchers and reduced researcher variation, thus ensuring greater validity of the data collection and results. The model was utilized both as a guide for the researchers in subsequent phases of the study and during the interviews.

### *3.2 Design and Planning*

In this phase, the detailed research procedures for the case study were designed and preparations were made for data collection. These prepara-

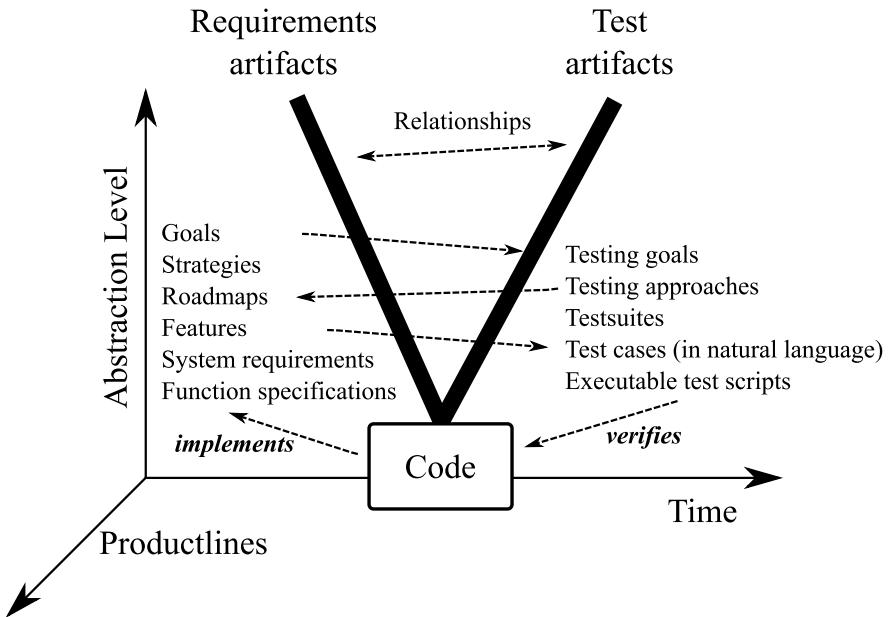


Figure 2.2: The conceptual model of the area under study, produced in phase 1

tions included designing the interview guide and selecting the cases and interviewees.

The interview guide was based on the research questions and aspects, and the conceptual model produced in the *Definition* phase (see Figures 2.1 and 2.2). The guide was constructed and refined several times by three researchers and reviewed by another four. User scenarios related to aligning requirements and testing, and examples of alignment metrics were included in the guide as a basis for discussions with the interviewees. The interview questions were mapped to the research questions to ensure that they were all covered. The guide was updated twice; after two pilot interviews, and after six initial interviews. Through these iterations the general content of the guide remained the same, though the structure and order of the interview questions were modified and improved. The resulting interview guide is published by Runeson et al. [2012, appendix C]. Furthermore, a consent information letter was prepared to make each interviewee aware of the conditions of the interviews and their rights to refuse to answer and to withdraw at any time. The consent letter is published by Runeson et al. [2012, appendix E].

The case selection was performed through a brainstorming session held within the group of researchers where companies and interviewee profiles that would match the research goals were discussed. In order to maximize

Table 2.1: Overview of the companies covered by the case study. At company F a major process change was taking place at the time of the study and data specific to the previous waterfall-based process are marked with “previous”.

| Company  | A                                       | B   | C   | D   | E  | F   |
|--|---|---|---|---|--|---|
| Type of company                                  | Software development, embedded products | Consulting  | Software development  | Systems engineering, embedded products                                | Software development, embedded products                            | Software development, embedded products                               |
| # employees in software development organization | 125–150                                 | 135   | 500   | 50–100  | 300–350  | 10,000  |
| # employees in typical project                   | 10                                      | Mostly 4–10, but varies greatly                     | 50–80   | software developers: 10–20  | 6–7 per team, 10–15 teams  | Previous process: 800–1,000 person years                              |
| Distributed                                      | No                                      | Co-located (per project; often on-site at customer) | Yes   | Yes   | Yes  | Yes   |
| Domain / System type                             | Computer networking equipment           | Advisory / technical services                       | Rail traffic management   | Automotive  | Telecom  | Telecom   |
| Source of requirements                           | Market driven                           | Bespoke   | Bespoke   | Bespoke and market driven   | Bespoke and market driven  | Bespoke and market driven   |
| Main quality focus                               | Availability, performance, security     | Depends on customer focus                           | Safety  | Safety  | Availability, performance, reliability, security                   | Performance, stability  |
| Certification                                    | No software related certification       | No  | ISO9001, ISO14001, OHSAS18001   | ISO9001, ISO14001   | ISO9001, ISO14001 (aiming towards adhering to TL9000)              | ISO9001   |
| Process Model                                    | Iterative                               | Agile in variants                                   | Waterfall   | RUP, Scrum  | Scrum, eRUP, a sprint is 3 months                                  | Iterative with gate decisions (agile influenced), previous: Waterfall |
| Duration of a typical project                    | 6–8 months                              | No typical project                                  | 1–5 years to first delivery, then new software release for 1–10 years | 1–5 years to first delivery, then new software release for 1–10 years | 1 year   | Previous process: 2 years   |
| # requirements in typical project                | 100 (20–30 pages HTML)                  | No typical project                                  | 600–800 at system level   | For software: 20–40 use cases   | 500–700 user stories   | Previous process: 14,000  |
| # of test cases in typical project               | ~1,000 test cases                       | No typical project                                  | 250 at system level   | 11000+  | Previous process: 200,000 at platform level, 1,000 at system level |   |
| Product Lines                                    | Yes                                     | No  | Yes   | Yes   | Yes  | Yes (with new agile process model)                                    |
| Open Source                                      | Yes                                     | Yes. Wide use including contributions               | Yes, partly   | No  | No   | Yes (with new agile process model)                                    |

the variation of companies selected from the industrial collaboration network, with respect to size, type of process, application domain and type of product, a combination of maximum variation selection and convenience selection was applied [Runeson et al., 2012, p. 35, 112]. The characteristics of the case companies are briefly summarized in Table 2.1. It is clear from the summary that they represent: a wide range of domains; size from 50 to 1,000 software developers; bespoke and market driven development; waterfall and iterative processes; using open source components or not, etc. At the time of the interviews a major shift in process model, from waterfall to agile, was underway at company F. Hence, for some affected factors in Table 2.1, information is given as to for which model the data is valid.

Our aim was to cover processes and artifacts relevant to REST alignment for the whole life cycle from requirements definition through development to system testing and maintenance. For this reason, interviewees were selected to represent the relevant range of viewpoints from requirements to testing, both at managerial and at engineering level. Initially, the company contact persons helped us find suitable people to interview. This was complemented by snowball sampling [Robson, 2002] by asking the interviewees if they could recommend a person or a role in the company whom we could interview in order to get alignment-related information. These suggestions were then matched against our aim to select interviewees in order to obtain a wide coverage of the processes and artifacts of interest. The selected interviewees represent a variety of roles, working with requirements, testing and development; both engineers and managers were interviewed. The number of interviews per company was selected to allow for going in-depth in one company (company F) through numerous interviews. Additionally, for this large company the aim was to capture a wide view of the situation and thus mitigate the risk of a skewed sample. For the other companies, three interviews were held per company. An overview of the interviewees, their roles and level of experience is given in Table 2.2. Note that for company B, the consultants that were interviewed typically take on a multitude of roles within a project even though they can mainly be characterized as software developers they also take part in requirements analysis and specification, design and testing activities.

### 3.3 Evidence Collection

A semi-structured interview strategy [Robson, 2002] was used for the interviews, which were performed over a period of 1 year starting in May 2009. The interview guide [Runeson et al., 2012, appendix C] acted as a checklist to ensure that all selected topics were covered. Interviews lasted for about 90 minutes. Two or three researchers were present at each interview, except for five interviews, which were performed by only one researcher. One of

Table 2.2: Overview of interviewees' roles at their companies incl. level of experience in that role; S(enior) = more than 3 years, or J(unior) = up to 3 years. Xn refers to interviewee n at company X. Note: most interviewees have additional previous experience

| Role                  | A      | B                            | C                 | D                    | E                          | F  |
|-----------------------|--------|------------------------------|-------------------|----------------------|----------------------------|--|
| Requirements engineer |        |                              |                   |                      | F1 (S), F6 (S), F7 (S)     |  |
| Systems architect     |        |                              |                   | D3 (J) E1 (S) F4 (S) |                            |  |
| Software developer    |        | B1 (J),<br>BS (S),<br>B3 (S) |                   |                      | F13 (S)                    |  |
| Test engineer         | A2 (S) |                              | C1 (S),<br>C2 (J) | D2 (S)               | E3 (S)                     | F9 (S), F10 (S),<br>F11 (J), F12 (S),<br>F14 (S) |
| Project manager       | A1 (J) |                              | C3 (S)            | D1 (S)               |                            | F3 (J), F8 (S)                                   |
| Product manager       | A3 (S) |                              |                   |                      | E2 (S)                     |  |
| Process manager       |        |                              |                   |                      | F2 (J), F5 (S),<br>F15 (J) |  |

the interviewers led the interview, while the others took notes and asked additional questions for completeness or clarification. After consent was given by the interviewee audio recordings were made of each interview. All interviewees consented.

The audio recordings were transcribed word by word and the transcriptions were validated in two steps to eliminate unclarities and misunderstandings. These steps were: (i) another researcher, primarily one who was present at the interview, reviewed the transcript, and (ii) the transcript was sent to the interviewee with sections for clarification highlighted and the interviewee had a chance to edit the transcript to correct errors or explain what they meant. These modifications were included into the final version of the transcript, which was used for further data analysis.

The transcripts were divided into chunks of text consisting of a couple of sentences each to enable referencing specific parts of the interviews. Furthermore, an anonymous code was assigned to each interview and the names of the interviewees were removed from the transcripts before data analysis in order to ensure anonymity of the interviewees.

### 3.4 Data Analysis

Once the data was collected through the interviews and transcribed (see Figure 2.1), a three-stage analysis process was performed consisting of:

coding, abstraction and grouping, and interpretation. These multiple steps were required to enable the researchers to efficiently navigate and consistently interpret the huge amounts of qualitative data collected, comprising more than 300 pages of interview transcripts.

*Coding* of the transcripts, i. e. the chunks, was performed to enable locating relevant parts of the large amounts of interview data during analysis. A set of codes, or keywords, based on the research and interview questions was produced, initially at a workshop with the participating researchers. This set was then iteratively updated after exploratory coding and further discussions. In the final version, the codes were grouped into multiple categories at different abstraction levels, and a coding guide was developed. To validate that the researchers performed coding uniformly, one interview transcript was selected and coded by all researchers. The differences in coding were then discussed at a workshop and the coding guide was subsequently improved. The final set of codes was applied to all the transcripts. The coding guide and some coding examples are published by Runeson et al. [2012, appendix D].

*Abstraction and grouping* of the collected data into statements relevant to the goals and questions for our study was performed in order to obtain a manageable set of data that could more easily be navigated and analyzed. The statements can be seen as an index, or common categorization of sections belonging together, in essence a summary of them as done by Gorschek and Wohlin [2006], Gorschek and Davis [2008], Pettersson et al. [2008], and Höst et al. [2010]. The statements were each given a unique identifier, title and description. Their relationship to other statements, as derived from the transcripts, was also abstracted. The statements and relationships between them were represented by nodes connected by directional edges. Figure 2.3 shows an example of the representation designed and used for this study. In particular, the figure shows the abstraction of the interview data around cross-role reviews of requirements, represented by node N4. For example, the statement 'cross-role reviews' was found to contribute to statements related to requirements quality. Each statement is represented by a node. For example, N4 for 'cross-role review', and N1, N196 and N275 for the statements related to requirements quality. The connections between these statements are represented by a 'contributes to' relationship from N4 to each of N1, N196 and N275. These connections are denoted by a directional edge tagged with the type of relationship. For example, the tags 'C' for 'contributes to', 'P' for 'prerequisite for' and 'DC' for 'does not contribute to'. In addition, negation of one or both of the statements can be denoted by applying a post- or prefix 'not' (N) to the connection. The type of relationships used for modeling the connections between statements were discussed, defined and agreed on in a series of work meetings. Traceability to the origin of the statements and the relationships between them was captured and maintained by noting the id of the

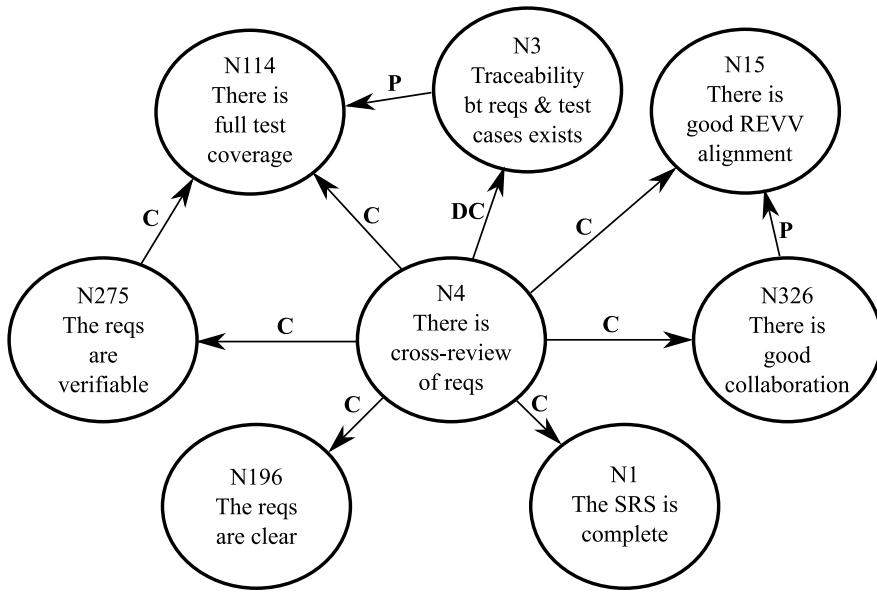


Figure 2.3: Part of the abstraction representing the interpretation of the interviewee data. The relationships shown denote C—contribute to, P—prerequisite for, and DC—does not contribute to

relevant source chunk, both for nodes and for edges. This is not shown in Figure 2.3.

The identified statements including relationships to other statements were extracted per transcript by one researcher per interview. To ensure a consistent abstraction among the group of researchers and to enhance completeness and correctness, the abstraction for each interview was reviewed by at least one other researcher and agreed after discussing differences of opinion. The nodes and edges identified by each researcher were merged into one common graph consisting of 341 nodes and 552 edges.

*Interpretation* of the collected evidence involved identifying the parts of the data relevant to a specific research question. The abstracted statements derived in the previous step acted as an index into the interview data and allowed the researchers to identify statements relevant to the research questions of challenges and practices. This interpretation of the interview data was performed by analyzing a graphical representation of the abstracted statements including the connections between them. Through the analysis nodes and clusters of nodes related to the research questions were identified. This is similar to explorative coding and the identified codes or clusters represented REST alignment challenges and practices with one cluster (code) per challenge and per practice. Due to the large amount of data, the

analysis and clustering was initially performed on sub-sets of the graphical representation, one for each company. The identified clusters were then iteratively merged into a common set of clusters for the interviews for all companies. For example, for the nodes shown in Figure 2.3 the statements 'The requirements are clear' (N196) and 'The requirements are verifiable' (N275) were clustered together into the challenge 'Defining clear and verifiable requirements' (challenge Ch3.2, see Section 4.1) based on connections (not shown in the example) to other statements reflecting that this leads to weak alignment.

Even with the abstracted representation of the interview transcripts, the interpretation step is a non-trivial task which requires careful and skillful consideration to identify the nodes relevant to specific research questions. For this reason, the clustering that was performed by Bjarnason was reviewed and agreed with Runeson. Furthermore, the remaining unclustered nodes were reviewed by Engström, and either mapped to existing clusters, suggested for new clusters or judged to be out of scope for the specific research questions. This mapping was then reviewed and agreed with Bjarnason.

Finally, the agreed clusters were used as an index to locate the relevant parts of the interview transcripts (through traces from the nodes and edges of each cluster to the chunks of text). For each identified challenge and practice, and mapping between them, the located parts of the transcriptions were then analyzed and interpreted, and reported in Sections 4.1, 4.2 and 4.3, respectively for challenges, practices, and the mapping.

### 3.5 Threats to Validity

There are limitations and threats to the validity to all empirical studies, and so also for this case study. As suggested by Runeson and Höst [2009], the construct validity, external validity and reliability were analyzed in the phases leading up to the analysis phase of the case study, see Figure 2.1. We also report measures taken to improve the validity of the study.

#### 3.5.1 Construct Validity

Construct validity refers to how well the chosen research method has captured the concepts under study. There is a risk that academic researchers and industry practitioners may use different terms and have different frames of reference, both between and within these categories of people. In addition, the presence of researchers may threaten the interviewees and lead them to respond according to assumed expectations. The selection of interviewees may also give a limited or unbalanced view of the construct. In order to mitigate these risks, we took the following actions in the design step:

- *Design of the interview guide and reference model.* The interview guide was designed based on the research questions and reviewed for completeness and consistency by other researchers. It was piloted during two interviews and then revised again after another six. The risk that the language and terms used may not be uniformly understood was addressed by producing a conceptual model (see Figure 2.2), which was shown to the interviewees to explain the terminology. However, due to the semi-structured nature of the guide and the different interviewers involved the absence of interviewee data for a certain concept, challenge or practice cannot be interpreted as the absence of this item either in the interviewees experience or in the company. For similar reasons, the results do not include any ranking or prioritization as to which challenges and practices are the most frequent or most effective.
- *Prolonged involvement.* The companies were selected so that at least one of the researchers had a long-term relation with them. This relationship helped provide the trust needed for openness and honesty in the interviews. To mitigate the bias of knowing the company too well, all but five interviews (companies D and E) were conducted by more than one interviewer.
- *Selection of interviewees.* To obtain a good representation of different aspects, a range of roles were selected to cover requirement, development and testing, and also engineers as well as managers, as reported in Table 2.2. The aim was to cover the relevant aspects described in the conceptual model, produced during the Definition phase (see Figure 2.2). There is a risk that the results might be biased due to a majority of the interviewees being from Company F. However, the results indicate that this risk was minor, since a majority of the identified items (see Section 4) could be connected to multiple companies.
- *Reactive bias:* The presence of a researcher might limit or influence the outcome either by hiding facts or responding after assumed expectations. To reduce this threat the interviewees were guaranteed anonymity both within the company and externally. In addition, they were not given any rewards for their participation and had the right to withdraw at any time without requiring an explanation, though no interviewees did withdraw. This approach indicated that we were interested in obtaining a true image of their reality and encouraged the interviewees to share this.

### 3.5.2 Internal Validity

Even though the conclusions in this chapter are not primarily about causal relations, the identification of challenges and practices somewhat resembles identifying factors in causal relations. In order to mitigate the risk

of identifying incorrect factors, we used data source triangulation by interviewing multiple roles at a company. Furthermore, extensive observer triangulation was applied in the analysis by always including more than one researcher in each step. This strategy also partly addressed the risk of incorrect generalizations when abstracting challenges and practices for the whole set of companies. However, the presented results represent one possible categorization of the identified challenges and practices. This is partly illustrated by the fact that not all identified practices can be connected to a challenge.

The interviews at one of the case companies were complicated by a major process change that was underway at the time of the study. This change posed a risk of confusing the context for which a statement had been experienced; the previous (old) way of working or the newly introduced agile practices. To mitigate this risk, we ensured that we correctly understood which process the response concerned, i. e. the previous or the current process.

Furthermore, due to the nature of semi-structured interviews in combination with several interviewers it is likely that different follow-on questions were explored by the various researchers. This risk was partly mitigated by jointly defining the conceptual model and agreeing on a common interview guide that was used for all interviews. However, the fact remains that there are differences in the detailed avenues of questioning which has resulted in only being able to draw conclusions concerning what was actually said at the interviews. So, for example, if the completeness of the requirements specification (Ch3.2) was not explicitly discussed at an interview no conclusions can be drawn concerning if this is a challenge or not for that specific case.

### 3.5.3 *External Validity*

For a qualitative study like this, external validity can never be assured by sampling logic and statistical generalization, but by analytical generalization which enables drawing conclusions and, under certain conditions, relating them also to other cases [Robson, 2002, Runeson et al., 2012]. This implies that the context of the study must be compared to the context of interest for the findings to be generalized to. To enable this process, we report the characteristics of the companies in as much detail as possible considering confidentiality (see Table 2.1). The fact that six different companies of varying size and domain are covered by the study, and some results are connected to the variations between them indicates that the results are more general than if only one company had been studied. But, of course, the world consists of more than six kinds of companies, and any application of the results of this study need to be mindfully tailored to other contexts.

### 3.5.4 Reliability

The reliability of the study relates to whether the same outcome could be expected with another set of researchers. For qualitative data and analysis, which are less procedural than quantitative methods, exact replication is not probable. The analysis lies in interpretation and coding of words, and the set of codes would probably be partly different with a different set of researchers.

To increase the reliability of this study and to reduce the influence by single researchers, several researchers have taken part in the study in different roles. All findings and each step of analysis have been reviewed by and agreed with at least one other researcher. In addition, a systematic and documented research process has been applied (see Figure 2.1) and a trace of evidence has been retained for each analysis steps. The traceability back to each source of evidence is documented and kept even in this report to enable external assessment of the chain of evidence, if confidentiality agreements would allow.

Finally, the presentation of the findings could vary depending on categorization of the items partly due to variation in views and experience of individual researchers. For example, a challenge in achieving alignment such as Ch2 *Collaborating successfully* (see Section 4.1.2) could be identified also as a practice at the general level, e.g. to collaborate successfully could be defined as an alignment practice. However, we have chosen to report specific practices that may improve collaboration and thereby REST alignment. For example, P1.1 *Customer communication at all requirements levels and phases* can support improved coordination of requirements between the customer and the development team. To reduce the risk of bias in this aspect, the results and the categorization of them was first proposed by one researcher and then reviewed by four other researchers leading to modifications and adjustments.

## 4 RESULTS

Practitioners from all six companies in the study found alignment of RE with ST to be an important, but challenging, factor in developing products. REST alignment was seen to affect the whole project life cycle, from the contact with the customer and throughout software development. The interviewees stated clearly that good alignment is essential to enable smooth and efficient software development. It was also seen as an important contributing factor in producing software that meets the needs and expectations of the customers. A software developer stated that alignment is “very important in creating the right system” (B1:27<sup>2</sup>). One interviewee described the

---

<sup>2</sup> Reference to source is given by interviewee code, see Table 2.2.

customer's view of a product developed with misaligned requirements as: "There wasn't a bug, but the behavior of the functionality was interpreted or implemented in such a way that it was hard to do what the customer [originally] intended" (A3:43). Another interviewee mentioned that alignment between requirements and verification builds customer trust in the end product since good alignment allows the company to "look into the customer's eyes and explain what have we tested... on which requirements" (D2:10).

In general, the interviewees expressed that weak and unaligned communication of the requirements often cause inconsistencies that affect the verification effort. A common view was that these inconsistencies, caused by requirements that are misunderstood, incorrect or changed, or even not communicated, leads to additional work in updating and re-executing test cases. Improved alignment, on the other hand, was seen to make "communication between different levels in the V-model a lot easier" (E3:93). One of the interviewed testers stated: "Alignment is necessary. Without it we [testers] couldn't do our job at all" (C1:77).

Below, we present the results concerning the challenges of alignment (Ch1–Ch10) and the practices (P1–P10) used, or suggested, by the case companies to address REST challenges. Table 2.3 provides an overview of the challenges found for each company, while Table 2.4 contains an overview of the practices. Table 2.6 shows which challenges each practices is seen to address.

#### 4.1 Alignment Challenges

The alignment challenges identified through this study are summarized in Table 2.3. Some items have been categorized together as one challenge, resulting in 10 main challenges where some consist of several related challenges. For example, Ch3 *Requirements specification quality* consists of three challenges (Ch3.1–Ch3.3) concerning different aspects of requirements quality. Each challenge including sub items is described in the subsections that follow.

##### 4.1.1 Challenge 1: Aligning Goals and Perspectives within an Organization (Ch1)

The alignment of goals throughout the organization was mentioned by many interviewees as vital in enabling cooperation among different organizational units (see challenge 2 in Section 4.1.2). However, goals were often felt to be missing or unclearly defined, which could result in "making it difficult to test [the goals]" (B3:17). In several companies problems with differing and unaligned goals were seen to affect the synchronization between requirements and testing, and cause organizational units to

Table 2.3: Alignment challenges mentioned for each company.

| Category              | Id    | Challenge   | Company <sup>1</sup> |   |   |   |   |   |
|-----------------------|-------|---|----------------------|---|---|---|---|---|
|                       |       |   | A                    | B | C | D | E | F |
| Req spec quality      | Ch1   | Aligning goals and perspectives within an organization                | X                    | X | X | X | X | X |
|                       | Ch2   | Cooperating successfully  |                      | X | X | X | X | X |
|                       | Ch3.1 | Defining clear and verifiable requirements                            |                      |   | X | X | X | X |
|                       | Ch3.2 | Defining complete requirements  |                      | X | X | X | X | X |
|                       | Ch3.3 | Keeping requirements documents updated                                |                      |   |   |   | X | X |
| ST quality            | Ch4.1 | Full test coverage  |                      | X | X | X | X | X |
|                       | Ch4.2 | Defining a good verification process                                  |                      |   |   |   | X | X |
|                       | Ch4.3 | Verifying quality requirements  |                      | X | X |   | X | X |
|                       | Ch5   | Maintaining alignment when requirements change                        | X                    | X |   |   | X | X |
| Req's abstract levels | Ch6.1 | Defining requirements at abstraction level well matched to test cases |                      |   |   | X | X | X |
|                       | Ch6.2 | Coordinating requirements at different abstraction levels             | X                    |   |   |   | X | X |
| Traceability          | Ch7.1 | Tracing between requirements and test cases                           | X                    | X | X | X | X | X |
|                       | Ch7.2 | Tracing between requirements abstraction levels                       |                      |   | X | X | X | X |
|                       | Ch8   | Time and resource availability  |                      |   | X | X | X | X |
|                       | Ch9   | Managing a large document space                                       |                      | X | X |   | X | X |
|                       | Ch10  | Outsourcing of components or testing                                  |                      |   | X | X |   | X |

<sup>1</sup> A blank cell means that the challenge was not mentioned during the interviews, not that it is not experienced.

counteract each other in joint development projects. For example, a product manager mentioned that at times, requirement changes needed from a business perspective conflicted with the goals of the development units; “They [business roles] have their own directives and … schedule target goals” and “they can look back and see which product was late and which product was good” (A3:74). In other words, misaligned goals may have an impact on both time schedules and product quality.

Many interviewees described how awareness and understanding of different perspectives on the problem domain is connected to better communication and cooperation, both towards the customers and external suppliers, and internally between competence areas and units. When there is a lack of aligned perspectives, the customer and the supplier often do not have the same understanding of the requirements. This may result in “errors in misunderstanding the requirements” (B3:70). Lack of insight into and awareness of different perspectives was also seen to result in decisions (often made by other units) being questioned and requirements changed at a late stage in the development cycle with a subsequent increase in cost and risk. For example, a systems architect described that in a project where there is a “higher expectations on the product than we [systems architect] scoped into it” (E1:20) a lot of issues and change requests surface in the late project phases. A software developer stated concerning the communication between requirements engineers and developers that “if both have a common perspective [of technical possibilities], then it would be easier to understand what [requirements] can be set and what cannot be set” (F13:29). Or in other words, with an increased common understanding technically infeasible requirements can be avoided already at an early stage.

Weak alignment of goals and perspectives implies a weak coordination at higher organizational levels and that strategies and processes are not synchronized. As stated by a process manager, the involvement of many separate parts of an organization then leads to “misunderstandings and misconceptions and the use of different vocabulary” (F2:57). In addition, a test engineer at Company A mentioned that for the higher abstraction levels there were no attempts to synchronize, for example, the testing strategy with the goals of development projects to agree on important areas to focus on (A2:105). Low maturity of the organization was thought to contribute to this and result in the final product having a low degree of correspondence to the high-level project goals. A test engineer said: “In the long run, we would like to get to the point where this [product requirements level] is aligned with this [testing activities]” (A2:119).

#### *4.1.2 Challenge 2: Cooperating Successfully (Ch2)*

All the companies included in our study described close cooperation between roles and organizational units as vital for good alignment and coordination of both people and artifacts. Weak cooperation is experienced to negatively affect the alignment, in particular at the product level. A product manager stated that “an ‘us and them’ validation of product level requirements is a big problem” (A3:058–059). Ensuring clear agreement and communication concerning which requirements to support is an important collaboration aspect for the validation. At Company F (F12:063) lack of cooperation in the early phases in validating requirements has been experienced to result in late discovery of failures in meeting important product requirements. The development project then say at a late stage: “We did not approve these requirements, we can’t solve it” (F12:63) with the consequence that the requirements analysis has to be re-done. For Company B (consulting in different organizations) cooperation and communication was even described as being prioritized above formal documentation and processes, expressed as: “We have succeeded with mapping requirements to tests since our process is more of a discussion” (B3:49). Several interviewees described that alignment at product and system level, in particular, is affected by how well people cooperate (C2:17, E1:44, 48, E2:48, F4:66, F15:46). When testers have a good cooperation and frequently communicate with both requirements-related and development-related roles, this leads to increased alignment (E3:093).

Organizational boundaries were mentioned as further complicating and hindering cooperation between people for two of the companies, namely companies E and F. In these cases, separate organizational units exist for requirements (E2:29, E3:94, F2:119), usability (F10:108) and testing (F3:184). As one interviewee said: “it is totally different organizations, which results in ... misunderstandings and misconceptions ... we use different words” (F02:57). Low awareness of the responsibilities and tasks of different organizational units was also claimed to negatively affect alignment (F2:264). This may result in increased lead times (E1:044, F15:033), need for additional rework (E1:150, E1:152), and conflicts in resource allocation between projects (F10:109, E1:34).

#### *4.1.3 Challenge 3: Good Requirements Specification Quality (Ch3)*

“If we don’t have good requirements the tests will not be that good” (D3:14). When the requirement specification is lacking the testers need to guess and make up the missing information since “the requirements are not enough for writing the software and testing the software” (D3:19). This both increases the effort required for testing and the risk of misinterpretation and missing vital customer requirements. One process manager ex-

pressed that the testability of requirements can be improved by involving testers and that “one main benefit [of alignment] is improving the requirements specifications” (F2:62). A test leader at the same company identified that a well aligned requirements specification (through clear agreement between roles and tracing between artifacts) had positive effects such as “it was very easy to report when we found defects, and there were not a lot of discussions between testers and developers, because everyone knew what was expected” (F9:11).

There are several aspects to good requirements that were found to relate to alignment. In the study, practitioners mentioned good requirements as being verifiable, clear, complete, at the right level of abstraction, and up-to-date. Each aspect is addressed below.

**DEFINING CLEAR AND VERIFIABLE REQUIREMENTS (CH3.1)** was mentioned as a major challenge in enabling good alignment of requirements and testing, both at product and at detailed level. This was mentioned for four of the six companies covered by our study, see Table 2.3. Unclear and non-verifiable requirements were seen as resulting in increased lead times and additional work in later phases in clarifying and redoing work based on unclear requirements (F2:64, D1:80). One test manager said that “in the beginning the requirements are very fuzzy. So it takes time. And sometimes they are not happy with our implementation, and we have to do it again and iterate until it’s ready” (F11:27, similar in E3:44). Failure to address this challenge ultimately results in failure to meet the customer expectations with the final product. A project manager from company D expressed this by saying that non-verifiable requirements is the reason “why so many companies, developers and teams have problems with developing customer-correct software” (D1:36).

**DEFINING COMPLETE REQUIREMENTS (CH3.2)** was claimed to be required for successful alignment by interviewees from four companies, namely companies B, D, E and F. As expressed by a systems architect from Company D, “the problem for us right now is not [alignment] between requirements and testing, but that the requirements are not correct and complete all the time” (D3:118). Complete requirements support achieving full test coverage to ensure that the full functionality and quality aspects are verified (F14:31). When testers are required to work with incomplete requirements, additional information is acquired from other sources, which requires additional time and effort to locate (D3:19).

**KEEPING REQUIREMENTS DOCUMENTATION UPDATED (CH3.3)**. Several interviewees from company F described how a high frequency of change leads to the requirements documentation not being kept updated, and consequently the documentation cannot be relied on (F14:44, F5:88).

When a test for a requirement then fails, the first reaction is not: "this is an error", but rather "is this really a relevant requirement or should we change it" (F5:81). Mentioned consequences of this include additional work to locate and agree to the correct version of requirements and re-work (F3:168) when incorrect requirements have been used for testing. Two sources of requirements changes were mentioned, namely requested changes that are formally approved (F14:50), but also changes that occur as the development process progresses (during design, development etc.) that are not raised as formal change requests (F5:82, F5:91, F11:38). When the requirements documentation is not reliable, the projects depend on individuals for correct requirements information. As expressed by one requirements engineer: "when you lose the people who have been involved, it is tough. And, things then take more time" (F1:137).

#### 4.1.4 Challenge 4: Validation and Verification Quality

Several issues with validation and verification were mentioned as alignment challenges that affect the efficiency and effectiveness of the testing effort. One process manager with long experience as a tester said: "We can run 100,000 test cases but only 9% of them are relevant" (F15:152). Testing issues mentioned as affecting alignment were: obtaining full test coverage, having a formally defined verification process and the verification of quality requirements.

**FULL TEST COVERAGE (CH4.1).** Several interviewees described full test coverage of the requirements as an important aspect of ensuring that the final product fulfills the requirements and the expectations of the customers. As one software developer said: "having full test coverage with unit tests gives a better security ... check that I have interpreted things correctly with acceptance tests" (B1:117). However, as a project manager from Company C said: "it is very hard to test everything, to think about all the complexities" (C3:15). *Unclear* (Ch3.2, C1:4) and *non-verifiable requirements* (Ch3.1, A1:55, D1:78, E1:65) were mentioned as contributing to difficulties in achieving full test coverage of requirements for companies A, B, D and E. For certain requirements that are expressed in a verifiable way a project manager mentioned that they cannot be tested due to limitations in the process, competence and test tools and environments (A1:56). To ensure full test coverage of requirements the testers need knowledge of the full set of requirements, which is impeded in the case of *incomplete requirements specifications* (Ch3.3) where features and functionality are not described (D3:16). This can also be the case for requirements defined at a higher abstraction level (F2:211, F14:056). Lack of traceability between requirements and test cases was stated to making it harder to know when full test coverage has been obtained (A1:42). For company C, traceability was stated as time con-

suming but necessary to ensure and demonstrate full test coverage, which is mandatory when producing safety-critical software (C1:6, C1:31). Furthermore, obtaining sufficient coverage of the requirements requires analysis of both the requirement and the connected test cases (C1:52, D3:84, F14:212). As one requirements engineer said, “a test case may cover part of a requirement, but not test the whole requirement” (F7:52). Late requirements changes was mentioned as a factor contributing to the challenge of full test coverage (C1:54, F7:51) due to the need to update the affected test cases, which is hampered by failure to keep the *requirements specification updated after changes* (Ch3.3, A2:72, F15:152).

**HAVING A VERIFICATION PROCESS (CH4.2)** was mentioned as directly connected to good alignment between requirements and test. At company F, the ongoing shift towards a more agile development process had resulted in the verification unit operating without a formal process (F15:21). Instead, each department and project “tries to work their own way … that turns out to not be so efficient” (F15:23), especially so in this large organization where many units and roles are involved from the initial requirements definition to the final verification and launch. Furthermore, one interviewee who was responsible for defining the new verification process (F15) said that “the hardest thing [with defining a process] is that there are so many managers … [that don’t] know what happens one level down”. In other words, a verification process that supports requirements-test alignment needs to be agreed with the whole organization and at all levels.

**VERIFYING QUALITY REQUIREMENTS (CH4.3)** was mentioned as a challenge for companies B, D and F. Company B has verification of quality in focus with continuous monitoring of quality levels in combination with frequent releases; “it is easy to prioritize performance optimization in the next production release” (B1:52). However, they do not work proactively with quality requirements. Even though they have (undocumented) high-level quality goals the testers are not asked to use them (B1:57, B2:98); “when it’s not a broken-down [quality] requirement, then it’s not a focus for us [test and development]” (B3:47). Company F does define formal quality requirements, but these are often not fully agreed with development (F12:61). Instead, when the specified quality levels are not reached, the requirements, rather than the implementation, are changed to match the current behavior, thus resigning from improving quality levels in the software. As one test engineer said: “We currently have 22 requirements, and they always fail, but we can’t fix it” (F12:61). Furthermore, defining verifiable quality requirements and test cases was mentioned as challenging, especially for usability requirements (D3:84, F10:119). Verification is then faced with the challenge of subjectively judging if a requirement is passed or failed (F2:46, F10:119). At company F, the new agile practices of

detailling requirements at the development level together with testers was believed to, at least partly, address this challenge (F12:65). Furthermore, additional complication is that some quality requirements can only be verified through analysis and not through functional tests (D3:84).

#### *4.1.5 Challenge 5: Maintaining Alignment when Requirements Change (Ch5)*

Most of the companies of our study face the challenge of maintaining alignment between requirements and tests as requirements change. This entails ensuring that both artifacts and tracing between them are updated consistently. Company B noted that the impact of changes is specifically challenging for test since test code is more sensitive to changes than requirements specifications. "That's clearly a challenge, because [the test code is] rigid, as you are exemplifying things in more detail. If you change something fundamental, there are many tests and requirements that need to be modified" (B3:72).

Loss of traces from test cases to requirements over time was also mentioned to cause problems. When test cases for which traces have been outdated or lost are questioned, then "we have no validity to refer to ... so we have to investigate" (A2:53). In company A, the connection between requirements and test cases are set up for each project (A2:71): "This is a document that dies with the project"; a practice found very inefficient. Other companies had varying ambitions of a continuous maintenance of alignment and traces between the artifacts. A key for maintaining alignment when requirements change is that the requirements are actively used. When this is not the case there is a need for obtaining requirements information from other sources. This imposes a risk that "a requirement may have changed, but the software developers are not aware of it" (D3:97).

Interviewees implicitly connected the traceability challenge to tools, although admitting that "a tool does not solve everything ... Somebody has to be responsible for maintaining it and to check all the links ... if the requirements change" (C3:053). With or without feasible tools, tracing also requires personal assistance. One test engineer said, "I go and talk to him and he points me towards somebody" (A2:195). Furthermore, the frequency of changes greatly affects the extent of this challenge and is an issue when trying to establish a base-lined version of the requirements. Company C has good tool support and traceability links, but require defined versions to relate changes to. In addition, they have a product line, which implies that the changes must also be coordinated between the platform (product line) and the applications (products) (C3:019, C3:039).

#### 4.1.6 *Challenge 6: Requirements Abstraction Levels (Ch6)*

REST alignment was described to be affected by the abstraction levels of the requirements for companies A, D and F. This includes the relationship to the abstraction levels of the test artifacts and ensuring consistency between requirements at different abstraction levels.

**DEFINING REQUIREMENTS AT ABSTRACTION LEVELS WELL-MATCHED TO TEST CASES (CH6.1)** supports defining test cases in line with the requirements and with a good coverage of them. This was mentioned for companies D and F. A specific case of this at company D is when the testers “don’t want to test the complete electronics and software system, but only one piece of the software” (D3:56). Since the requirements are specified at a higher abstraction level than the individual components, the requirements for this level then need to be identified elsewhere. Sources for information mentioned by the interviewees include the design specification, asking people or making up the missing requirements (D3:14). This is also an issue when retesting only parts of a system which are described by a high-level requirement to which many other test cases are also traced (D3:56). Furthermore, synchronizing the abstraction levels between requirements and test artifacts was mentioned to enhance coverage (F14:31).

**COORDINATING REQUIREMENTS AT DIFFERENT ABSTRACTION LEVELS (CH6.2)**. When breaking down the high-level requirements (such as goals and product concepts) into detailed requirements at system or component level was mentioned as a challenge by several companies. A product manager described that failure to coordinate the detailed requirements with the overall concepts could result in that “the intention that we wanted to fulfill is not solved even though all the requirements are delivered” (A3:39). On the other hand, interviewees also described that the high-level requirements were often vague at the beginning when “it is very difficult to see the whole picture” (F12:144) and that some features are “too complex to get everything right from the beginning” (A3:177).

#### 4.1.7 *Challenge 7: Tracing Between Artifacts (Ch7)*

This challenge covers the difficulties involved in tracing requirements to test cases, and vice versa, as well as, tracing between requirements at different abstraction levels. Specific tracing practices identified through our study are described in Sections 4.2.6 and 4.2.7.

**TRACING BETWEEN REQUIREMENTS AND TEST CASES (CH7.1)**. The most basic kind of traceability, referred to as “conceptual mapping” in

Company A (A2:102), is having a line of thought (not necessarily documented) from the requirements through to the defining and assessing of the test cases. This cannot be taken for granted. Lack of this basic level of tracing is largely due to weak awareness of the role requirements in the development process. As a requirements process engineer in Company F says, "One challenge is to get people to understand why requirements are important; to actually work with requirements, and not just go off and develop and do test cases which people usually like doing" (F5:13).

Tracing by using matrices to map between requirements and test cases is a major cost issue. A test architect at company F states, that "we don't want to do that one to one mapping all the way because that takes a lot of time and resources" (F10:258). Companies with customer or market demands on traceability, e.g. for safety critical systems (companies C and D), have full traceability in place though "there is a lot of administration in that, but it has to be done" (C1:06). However, for the other case companies in our study (B3:18, D3:45; E2:83; F01:57), it is a challenge to implement and maintain this support even though tracing is generally seen as supporting alignment. Company A says "in reality we don't have the connections" (A2:102) and for Company F "in most cases there is no connection between test cases and requirements" (F1:157). Furthermore, introducing traceability may be costly due to large legacies (F1:57) and maintaining traceability is costly. However, there is also a cost for lack of traceability. This was stated by a test engineer in Company F who commented on degrading traceability practices with "it was harder to find a requirement. And if you can't find a requirement, sometimes we end up in a phase where we start guessing" (F12:112).

Company E has previously had a tradition of "high requirements on the traceability on the products backwards to the requirements" (E2:83). However, this company foresees problems with the traceability when transitioning towards agile working practices, and using user stories instead of traditional requirements. A similar situation is described for Company F, where they attempt to solve this issue by making the test cases and requirements one; "in the new [agile] way of working we will have the test cases as the requirements" (F12:109).

Finally, traceability for quality (a.k.a. non-functional) requirements creates certain challenges, "for instance, for reliability requirement you might ... verify it using analysis" (D3:84) rather than testing. Consequently, there is no single test case to trace such a quality requirement to, instead verification outcome is provided through an analysis report. In addition, tracing between requirements and test cases is more difficult "the higher you get" (B3:20). If the requirements are at a high abstraction level, it is a challenge to define and trace test cases to cover the requirements.

## TRACING BETWEEN REQUIREMENTS ABSTRACTION LEVELS (Ch7.2).

Another dimension of traceability is vertical tracing between requirements at different abstraction levels. Company C operates with a detailed requirements specification, which for some parts consists of sub-system requirements specifications (C1:31). In this case, there are no special means for vertical traceability, but pointers in the text. It is similar in Company D, where a system architect states that “sometimes it’s not done on each individual requirement but only on maybe a heading level or something like that” (D3:45). Company F use a high-end requirements management tool, which according to the requirements engineer “can trace the requirement from top level to the lowest implementation level” (F7:50).

Company E has requirements specifications for different target groups, and hence different content; one market oriented, one product oriented, and one with technical details (E1:104). The interviewee describes tracing as a “synch activity” without specifying in more detail. Similarly, Company F has “road-maps” for the long term development strategy, and there is a loosely coupled “connection between the road-maps and the requirements” to balance the project scope against strategy and capacity (F11:50).

### 4.1.8 Challenge 8: Time and Resource Availability (Ch8)

In addition to the time consuming task of defining and maintaining traces (Ch7), further issues related to time and resources were brought forward in companies C, E and F. Without sufficient resources for validation and verification the amount of testing that can be performed is not sufficient for the demands on functionality and quality levels expected of the products. The challenge of planning for enough test resources is related to the alignment between the proposed requirements and the time and resources required to sufficiently test them. A requirements engineer states that “I would not imagine that those who are writing the requirements in anyway are considering the test implications or the test effort required to verify them” (F6:181). A test manager confirms this view (F14:188). It is not only a matter of the amount of resources, but also in which time frame they are available (E1:18). Furthermore, availability of all the necessary competences and skills within a team was also mentioned as an important aspect of ensuring alignment. A software developer phrased it: “If we have this kind of people, we can set up a team that can do that, and then the requirements would be produced properly and hopefully 100% achievable” (F13:149). In addition, experienced individuals were stated to contribute to strengthening the alignment between requirements and testing, by being “very good at knowing what needs to be tested and what has a lower priority” (C2:91), thereby increasing the test efficiency. In contrast, inexperienced testing teams were mentioned for Company C as contributing to weaker alignment towards the overall set of requirements including goals

and strategies since they “verify only the customer requirements, but sometimes we have hazards in the system which require the product to be tested in a better way” (C2:32–33).

#### 4.1.9 *Challenge 9: Managing a Large Document Space (Ch9)*

The main challenge regarding the information management problems lies in the sheer numbers. A test engineer at Company F estimates that they have accumulated 50,000 requirements in their database. In addition, they have “probably hundreds of thousands of test cases” (F2:34, F12:74). Another test engineer at the same company points out that this leads to information being redundant (F11:125), which consequently may lead to inconsistencies. A test engineer at Company D identifies the constant change of information as a challenge; they have difficulties to work against the same baseline (D2:16).

Another test engineer at Company F sees information management as a tool issue. He states that “the requirements tool we have at the moment is not easy to work with ... Even, if they find the requirements they are not sure they found the right version” (F9:81). In contrast, a test engineer at company C is satisfied with the ability to find information in the same tool (C2). A main difference is that at Company F, 20 times as many requirements are handled than at Company C.

The investment into introducing explicit links between a huge legacy of requirements and test cases is also put forward as a major challenge for companies A and F. In addition, connecting and integrating different tools was also mentioned as challenging due to separate responsibilities and competences for the two areas of requirements and testing (F5:95, 120).

#### 4.1.10 *Challenge 10: Outsourcing or Off-shoring of Components or Testing (Ch10)*

Outsourcing and off-shoring of component development and testing create challenges both in agreeing to which detailed requirements to implement and test, and in tracing between artifacts produced by different parties. Company D stresses that the timing of the outsourcing plays a role in the difficulties in tracing component requirement specifications to the internal requirements at the higher level; “I think that’s because these outsourcing deals often have to take place really early in the development” (D3:92). Company F also mentions the timing aspect for acquisition of hardware components; “it is a rather formal structured process, with well-defined deliverables that are slotted in time” (F6:21).

When testing is outsourced, the specification of what to test is central and related to the type of testing. The set-up may vary depending on competence or cultural differences etc. For example, company F experienced

that cultural aspects influence the required level of detail in the specification; “we [in Europe] might have three steps in our test cases, while the same test case with the same result, but produced in China, has eight steps at a more detailed level” (F15:179). A specification of what to test may be at a high level and based on a requirements specification from which the in-sourced party derives tests and executes. An alternative approach is when a detailed test specification is requested to be executed by the in-sourced party (F6:251–255).

## *4.2 Practices for Improved Alignment*

This study has identified 27 different alignment practices, grouped into 10 categories (Table 2.4). Most of the practices are applied at the case companies, though some are suggestions made by the interviewees. These categories and the practices are presented below and discussed and summarized in Section 5. In Section 4.3 they are mapped to the challenges that they are seen to address.

### *4.2.1 Requirements Engineering Practices*

Requirements engineering practices are at the core of aligning requirements and testing. This category of practices includes customer communication and involving development-near roles in the requirements process. The interviewees described close cooperation and team work as a way to improve RE practices (F12:146) and thereby the coordination with developers and testers and avoid a situation where product managers say “redo it’ when they see the final product” (F12:143).

CUSTOMER COMMUNICATION AT ALL LEVELS AND IN ALL PHASES OF DEVELOPMENT (P1.1) was mentioned as an alignment practice for all but one of the case companies. The communication may take the form of customer-supplier co-location; interaction with the customer based on executable software used for demonstrations or customer validation; or agreed acceptance criteria between customer and supplier. For the smaller companies, and especially those with bespoke requirements (companies B and C), this interaction is directly with a physical customer. In larger companies (companies E and F), and especially within market driven development, a customer proxy may be used instead of the real customer, since there is no assigned customer at the time of development or there is a large organizational distance to the customer. Company F assigns a person in each development team “responsible for the feature scope. That person is to be available all through development and to the validation of that feature” (F2:109). Furthermore, early discussions about product road-

Table 2.4: Alignment practices and categories, and case companies for which they were mentioned.

| Category     | Id   | Challenge  | Company <sup>1</sup> |   |   |   |   |   |
|--------------|------|--|----------------------|---|---|---|---|---|
|              |      |  | A                    | B | C | D | E | F |
| Requirements | P1.1 | Customer communication at all requirements levels and phases |                      | X | X | X | X | X |
|              | P1.2 | Development involved in detailing requirements               | X                    | X |   |   |   | X |
|              | P1.3 | Cross-role requirements reviews                              |                      | X | X | X | X | X |
|              | P1.4 | Requirements review responsibilities defined                 |                      |   |   |   | X | X |
|              | P1.5 | Subsystem expert involved in requirements definition         |                      |   |   | X |   | X |
|              | P1.6 | Documentation of requirement decision rationales             |                      |   |   | S | S |   |
| Validation   | P2.1 | Test cases reviewed against requirements                     |                      |   |   |   |   | X |
|              | P2.2 | Acceptance test cases defined by customer                    | X                    |   |   |   |   |   |
|              | P2.3 | Product manager reviews prototypes                           | X                    |   |   |   | X |   |
|              | P2.4 | Management base launch decision on test report               |                      |   |   |   |   | X |
|              | P2.5 | User / Customer testing                                      |                      | X | X | X | X |   |
| Verification | P3.1 | Early verification start                                     |                      |   |   |   | X | X |
|              | P3.2 | Independent testing  |                      | X | X | X |   |   |
|              | P3.3 | Testers re-use customer feedback from previous projects      |                      |   |   | X | X | X |
|              | P3.4 | Training off-shore testers                                   |                      |   |   | X |   |   |
| Change       | P4.1 | Process for requirements changes involving ST                | X                    | X | X | X |   |   |
|              | P4.2 | Product-line requirements practices                          | X                    |   | X |   |   |   |
|              | P5   | Process enforcement  |                      | X |   |   | S |   |
| Tracing      | P6.1 | Document-level traces  |                      | X |   |   |   |   |
|              | P6.2 | Requirements-test case traces                                |                      |   |   |   |   | X |
|              | P6.3 | Test cases as requirements                                   | X                    |   |   |   |   | X |
|              | P6.4 | Same abstraction levels for requirements and test spec       |                      |   | X | X |   |   |
|              | P7   | Traceability responsibility role                             |                      |   | X | X | X |   |
| Tools        | P8.1 | Tool support for requirements and testing                    | X                    | X | X | X | X |   |
|              | P8.2 | Tool support for requirements-test case tracing              | X                    | X | X | X | X |   |
|              | P9   | Alignment metrics, e. g. test coverage                       |                      | X | X | X | X |   |
|              | P10  | Job rotation   | S                    |   | S |   |   |   |

<sup>1</sup> Experienced practices are marked with X, while suggested practices are denoted with S. A blank cell means that the practice was not mentioned during the interviews. It does not mean that it is not applied at the company.

maps from a 4 to 5 year perspective are held with customers and key suppliers (F6:29) as an initial phase of the requirements process.

**INVOLVING DEVELOPERS AND TESTERS IN DETAILED REQUIREMENTS (P1.2)** is another practice, especially mentioned by companies A and F. A product manager has established this as a deliberate strategy by conveying the vision of the product to the engineers rather than detailed requirements: "I'm trying to be more conceptual in my ordering, trying to say what's important and the main behavior" (A3:51). The responsibility for detailing the specification then shifts to the development organization. However, if there is a weak awareness of the customer or market perspectives, this may be a risky practice as "some people will not [understand this] either because they [don't] have the background or understanding of how customers or end-users or system integrators think" (A3:47). Testers may be involved to ensure the testability of the requirements, or even specify requirements in the form of test cases. Company F was in the process of transferring from a requirements-driven organization to a design-driven one. Splitting up the (previous) centralized requirements department resulted in "requirements are vaguer now. So it's more up to the developers and the testers to make their own requirements" (F12:17). Close cooperation around requirements when working agile was mentioned as vital by a product manager from Company E: "Working agile requires that they [requirements, development, and test] are really involved [in requirements work] and not only review" (E2:083).

**CROSS-ROLE REQUIREMENTS REVIEWS (P1.3)** across requirements engineers and testers is another practice applied to ensure that requirements are understood and testable (A2:65,C3:69, F2:38, F7:7). The practical procedures for the reviews, however, tend to vary. Company A has an early review of requirements by testers while companies C and D review the requirements while creating the test cases. Different interviewees from companies E and F mentioned one or the other of these approaches; the process seems to prescribe cross-role reviews but process compliance varies. A test engineer said "[the requirements are] usually reviewed by the testers. It is what the process says" (F11:107). Most interviewees mention testers' reviews of requirements as a good practice that enhances both the communication and the quality of the requirements, thereby resulting in better alignment of the testing effort. Furthermore, this practice was described as enabling early identification of problems with the test specification avoiding (more expensive) problems later on (C2:62). A systems architect from Company F described that close collaboration between requirements and testing around quality requirements had resulted in "one area where we have the best alignment" (F4:101).

**DEFINING A REQUIREMENTS REVIEW RESPONSIBLE** (P1.4) was mentioned as a practice that ensures that requirement reviews are performed (E2:18, F2:114). In addition, for Company F this role was also mentioned as reviewing the quality of the requirements specification (F2:114) and thereby directly addressing the alignment challenge of low quality of the requirements specification (Ch3).

**INVOLVING DOMAIN EXPERTS IN THE REQUIREMENTS DEFINITION** (P1.5) was mentioned as a practice to achieve better synchronization between the requirements and the system capabilities, and thereby support defining more realistic requirements. The expert “will know if we understand [the requirement] correctly or not” (D3:38), said a system architect. Similar to the previous RE practices, this practice was also mentioned as supporting alignment by enhancing the quality of the requirements (Ch3) which are the basis for software testing.

**DOCUMENTATION OF REQUIREMENT DECISION RATIONALES** (P1.6), and not just the current requirement version, was suggested as a practice that might facilitate alignment by interviewees from both of the larger companies in our study, namely E and F. “Softly communicating how we [requirements roles] were thinking” (E3:90) could enhance the synchronization between project phases by better supporting hand-over between the different roles (F4:39). In addition, the information could support testers in analyzing customer defect reports filed a long time after development was completed, and in identifying potential improvements (E3:90). However, the information needs to be easily available and connected to the relevant requirements and test cases for it to be practically useful to the testers (F1:120).

#### 4.2.2 *Validation Practices*

Practices for validating the system under development and ensuring that it is in-line with customer expectations and that the right product is built [[IEEE, 1990](#)] include test case reviews, automatic testing of acceptance test cases, and review of prototypes.

**TEST CASES ARE REVIEWED AGAINST REQUIREMENTS** (P2.1) at company F (F14:62). In their new (agile) development processes, the attributes of ISO9126 [[ISO/IEC, 2001](#)] are used as a checklist to ensure that not only functional requirements are addressed by the test cases, but also other quality attributes (F14:76).

**ACCEPTANCE TEST CASES DEFINED BY CUSTOMER** (P2.2), or by the business unit, is practiced at company B. The communication with the cus-

tomer proxy in terms of acceptance criteria for (previously agreed) user scenarios acts as a “validation that we [software developers] have interpreted the requirements correctly” (B1:117). This practice in combination with full unit test coverage of the code (B1:117) was experienced to address the challenge of achieving full test coverage of the requirements (Ch4, see Section 4.1 on page 60).

**REVIEWING PROTOTYPES** (P2.3) and GUI mock-ups was mentioned as an alignment practice applied at company A. With this practice, the product manager in the role as customer proxy validates that the developed product is in-line with the original product intents (A3:153,163). Company partners that develop tailor-made systems using their components may also be involved in these reviews.

**MANAGEMENT BASE LAUNCH DECISIONS ON TEST REPORTS** (P2.4) was mentioned as an important improvement in the agile way of working recently introduced at Company F. Actively involving management in project decisions and, specifically in deciding if product quality is sufficient for the intended customers was seen as ensuring and strengthening the coordination between customer and business requirements, and testing; “Management … have been moved down and [made to] sit at a level where they see what really happens” (F15:89).

**USER / CUSTOMER TESTING** (P2.5) is a practice emphasized by company B that apply agile development practices. At regular intervals, executable code is delivered, thus allowing the customer to test and validate the product and its progress (B3: 32, B3:99). This practice is also applied at company E, but with an organizational unit functioning as the user proxy (E3:22). For this practice to be effective the customer testing needs to be performed early on. This is illustrated by an example from company F, namely “before the product is launched the customer gets to test it more thoroughly. And they submit a lot of feedback. Most are defects, but there are a number of changes coming out of that. That’s very late in the process … a few weeks [...] before the product is supposed to be launched” (F1:12). If the feedback came earlier, it could be addressed, but not at this late stage.

#### 4.2.3 *Verification Practices*

Verification ensures that a developed system is built according to the specifications [IEEE, 1990]. Practices verifying that system properties are aligned to system requirements include starting verification early to allow time for feedback and change, using independent test teams, re-use of customer feedback obtained from previous projects, and training testers at outsourced or off-shored locations.

**EARLY VERIFICATION (P3.1)** is put forward as an important practice especially when specialized hardware development is involved, as for an embedded product. Verification is then initially performed on prototype hardware (F15:114). Since quality requirements mostly relate to complete system characteristics, early verification of these requirements is harder, but also more important. Company E states: "If we have performance issues or latency issues or database issues then we usually end up in weeks of debugging and checking and tuning" (E3:28).

**INDEPENDENT TEST TEAMS (P3.2)** are considered a good practice to reduce bias in interpreting requirements by ensuring that testers are not influenced by the developers' interpretation of requirements. However, this practice also increases the risk of misalignment when the requirements are insufficiently communicated since there is a narrower communication channel for requirements-related information. This practice was emphasized especially for companies with safety requirements in the transportation domain (companies C and D); "due to the fact that this is a fail-safe system, we need to have independence between testers and designers and implementers" (C3:24, similar in C2:39, D2:80), "otherwise they [test team] might be misled by the development team" (D1:41). Similarly, company F emphasizes alternative perspectives taken by an independent team. As a software developer said: "You must get another point of view of the software from someone who does not know the technical things about the in-depth of the code, and try to get an overview of how it works" (F13:32).

**TESTERS RE-USE CUSTOMER FEEDBACK FROM PREVIOUS PROJECTS (P3.3)** when planning the verification effort for later projects (F14:94), thereby increasing the test coverage. In addition to having knowledge of the market through customer feedback, verification organizations often analyze and test competitor products. With a stronger connection and co-ordination between the verification and business/requirements units, this information could be utilized in defining more accurate road-maps and product plans.

**TRAINING OFF-SHORE / OUTSOURCED TESTERS (P3.4)** in the company's work practices and tools increases the competence and motivation of the outsourced testers in the methods and techniques used by the outsourcing company. This was mentioned by a project manager from Company C as improving the quality of verification activities and the coordination of these activities with requirement (C3:49, 64).

#### 4.2.4 Change Management Practices

Practices managing the (inevitable) changes in software development may mitigate the challenge of maintaining alignment (Ch5, see Section 4.1.5). We identified practices related to the involvement of testing roles in the change management process and also practices connected to product lines as a means to support REST alignment.

**INVOLVING TESTING ROLES IN CHANGE MANAGEMENT (P4.1)**, in the decision making and in the communication of changes, is a practice mentioned by all companies, but one, as supporting alignment through increased communication and coordination of these changes with the test organization. “[Testers] had to show their impacts when we [product management] were deleting, adding or changing requirements” (E2:73) and “any change in requirement … means involving developer, tester, project manager, requirements engineer; sitting together when the change is agreed, so everybody is aware and should be able to update accordingly” (F8:25). In companies with formalized waterfall processes, a change control board (CCB) is a common practice for making decisions about changes. Company D has weekly meetings of the “change control board with the customer and we also have more internal change control boards” (D1:106). The transitioning to agile practices affected the change management process at companies E and F. At company F the change control board (CCB) was removed, thus enhancing local control at the expense of control of the whole development chain. As expressed by a process manager in company F: “I think it will be easy for developers to change it [the requirements] into what they want it to be” (F12:135). At company E centralized decisions were retained at the CCB (E2:73), resulting in a communication challenge; “sometimes they [test] don’t even know that we [product management] have deleted requirements until they receive them [as deleted from the updated specification]” (E2:73).

**PRODUCT-LINE REQUIREMENTS PRACTICES (P4.2)** are applied in order to reduce the impact of a requirements change. By sharing a common product line (a.k.a. platform), these companies separate between the requirements for the commonality and variability of their products. In order to reduce the impact of larger requirements changes and the risks these entail for current projects, company A “develop it [the new functionality] separately, and then put that into a platform” (A3:65). Company C uses product lines to leverage on invested test effort in many products. When changing the platform version “we need to do the impact analysis for how things will be affected. And then we do the regression test on a basic functionality to see that no new faults have been introduced” (C3:55).

#### *4.2.5 Process Enforcement Practices (P5)*

External requirements and regulations on certain practices affect the motivation and incentive for enforcing processes and practices that support alignment. This is especially clear in company C, which develops safety critical systems. “Since it is safety-critical systems, we have to show that we have covered all the requirements, that we have tested them” (C1:6). It is admitted that traceability is costly, but, non-negotiable in their case. “There is a lot of administration in that, in creating this matrix, but it has to be done. Since it is safety-critical systems, it is a reason for all the documentation involved” (C1:06). They also have an external assessor to validate that the processes are in place and are adhered to. An alternative enforcement practice was proposed by one interviewee from company F (which does not operate in a safety-critical domain) who suggested that alignment could be achieved by enforcing traceability through integrating process enforcement in the development tools (F14:161) though this had not been applied.

#### *4.2.6 Tracing Between Artifacts*

The tracing practices between requirements and test artifacts vary over a large range of options from simple mappings between documents to extensive traces between detailed requirements and test cases.

**DOCUMENT-LEVEL TRACES (P6.1)** where links are retained between related documents is the simplest tracing practice. This is applied at company A: “we have some mapping there, between the project test plan and the project requirement specification. But this is a fragile link” (A2:69).

**REQUIREMENT — TEST CASE TRACES (P6.2)** is the most commonly mentioned tracing practice where individual test cases are traced to individual requirements. This practice influences how test cases are specified: “It is about keeping the test case a bit less complex and that tends to lead to keep them to single requirements rather than to several requirements” (F6:123).

**USING TEST CASES AS REQUIREMENTS (P6.3)** where detailed requirements are documented as test cases is another option where the tracing become implicit at the detailed level when requirements and test cases are represented by the same entity. This practice was being introduced at company F. “At a certain level you write requirements, but then if you go into even more detail, what you are writing is probably very equivalent to a test case” (F5:113). While this resolves the need for creating and maintaining traces at that level, these test-case requirements need to be aligned to requirements and testing information at higher abstraction levels. “There

will be teams responsible for mapping these test cases with the high-level requirements" (F10:150). Company A has this practice in place, though not pre-planned but due to test cases being better maintained over time than requirements. "They know that this test case was created for this requirement some time ago [...] implicitly [...] the database of test cases becomes a requirements specification" (A2:51).

SAME ABSTRACTION LEVELS USED FOR REQUIREMENTS AND TEST SPECIFICATIONS (P6.4) is an alignment practice related to the structure of information. First, the requirements information is structured according to suitable categories. The requirements are then detailed and documented within each category, and the same categorization used for the test specifications. Company C has "different levels of requirements specifications and test specifications, top level, sub-system, module level, and down to code" (C3:67), and company D presents similar on the test processes and artifacts (D3:53). It is worth noting that both company C and D develop safety-critical systems. At company F, a project leader described "the correlation between the different test [levels]" and different requirement levels; at the most detailed level "test cases that specify how the code should work" and at the next level "scenario test cases" (F8:16).

#### *4.2.7 Practice of Traceability Responsible Role (P7)*

For large projects, and for safety-critical projects, the task of creating and maintaining the traces may be assigned to certain roles. In company E, one of the interviewees is responsible for consolidating the information from several projects to the main product level. "This is what I do, but since the product is so big, the actual checking in the system is done by the technical coordinator for every project" (E3:54). In one of the companies with safety-critical projects this role also exists; "a safety engineer [...] worked with the verification matrix and put in all the information [...] from the sub products tests in the tool and also we can have the verification matrix on our level" (C2:104).

#### *4.2.8 Tool Support*

Tool support is a popular topic on which everyone has an opinion when discussing alignment. The tool practices used for requirements and test management vary between companies, as does the tool support for tracing between these artifacts.

TOOL SUPPORT FOR REQUIREMENTS AND TEST MANAGEMENT (P8.1) varies hugely among the companies in this study, as summarized in Table 2.5. Company A uses a test management tool, while requirements are

Table 2.5: Tool usage for requirements and test cases, and for tracing between them.

|              | Requirements tool     | Tracing tool       | Testing tool                      |
|--------------|-----------------------|--------------------|-----------------------------------|
| Requirements | C, D, E, F (previous) |                    | F                                 |
| Traces       | C                     | D, E, F (previous) | F                                 |
| Test cases   | C                     |                    | A, D, E, F (current and previous) |

For company F the tool set-up prior to the major process change are also given (marked with 'previous').

stored as text. Companies D and E use a requirements management tool for requirements and a test management tool for testing. This was the previous practice at company F too. Company C uses a requirements management tool for both requirements and test, while Company F aims to start using a test management tool for both requirements and testing. Most of the companies use commercial tools, though company A has an in-house tool, which they describe as "a version handling system for test cases" (A2:208).

TOOL SUPPORT FOR REQUIREMENTS-TEST CASE TRACING (P8.2) is vital for supporting traceability between the requirements and test cases stored in the tools used for requirements and test management. Depending on the tool usage, tracing needs to be supported either within a tool, or two tools need to be integrated to allow tracing between them. For some companies, only manual tracing is supported. For example, at company D a systems architect describes that it is possible to "trace requirements between different tools such as [requirements] modules and Word documents" (D3:45). However, a software manager at the same company mentions problems in connecting the different tools and says "the tools are not connected. It's a manual step, so that's not good, but it works" (D1:111). Tracing within tools is practiced at company C where requirements and test cases are both stored in a commercial requirements management tool: "when we have created all the test cases for a certain release, then we can automatically make this matrix show the links between [system] requirements and test cases" (C1:8). Company F has used the between-tools practice "The requirements are synchronized over to where the test cases are stored" (F5:19). However, there are issues related to this practice. Many-to-many relationships are difficult to handle with the existing tool support (F2:167). Furthermore, relationships at the same level of detail are easier to handle than across different abstraction levels. One requirements engineer asks for "a tool that connects everything; your requirement with design documents with test cases with your code maybe even your planning document" (F5:17). In a large, complex system and its development organiza-

tion, there is a need for “mapping towards all kinds of directions – per function group, per test cases, and from the requirement level” (F11:139).

Many interviewees had complaints about their tools, and the integration between them. Merely having tool support in place is not sufficient, but it must be efficient and usable. For example, company E have tools for supporting traceability between requirements and test state of connected test cases but “we don’t do it because the tool we have is simply not efficient enough” (E3:57) to handle the test state for the huge amount of verified variants. Similarly, at company E the integration solution (involving a special module for integrating different tools) is no longer in use and they have reverted to manual tracing practices: “In some way we are doing it, but I think we are doing it manually in Excel sheets” (E2:49).

Finally, companies moving from waterfall processes towards agile practices tend to find their tool suite too heavyweight for the new situation (E3:89). Users of these tools not only include engineers, but also management, which implies different demands. A test manager states: “Things are easy to do if you have a lot of hands on experience with the tools but what you really need is something that the [higher level] managers can use” (F10:249).

#### 4.2.9 Alignment Metrics (P9)

Measurements can be used to gain control of the alignment between requirements and testing. The most commonly mentioned metrics concern test case coverage of requirements. For example, company C “measure[s] how many requirements are already covered with test cases and how many are not” (C1:64). These metrics are derived from the combined requirements and test management tool. Companies E and F have a similar approach, although with two different tools. They both point out that, in addition to the metrics, it is a matter of judgment to assess full requirements coverage. “If you have one requirement, that requirement may need 16 test cases to be fully compliant. But you implement only 14 out of those. And we don’t have any system to see that these 2 are missing” (E3:81) And, “just because there are 10 test cases, we don’t know if [the requirement] is fully covered” (F11:34). Furthermore, there is a versioning issue to be taken into account when assessing the requirements coverage for verification. “It is hard to say if it [coverage] should be on the latest software [version] before delivery or . . .?” (F10:224). The reverse relationship of requirements coverage of all test cases is not always in place or measured. “Sometimes we have test cases testing functionality not specified in the requirements database” (F11:133). Other alignment metrics were mentioned, for example, missing links between requirements and tests, number of requirements at different levels (F5:112), test costs for changed requirements (F14:205), and requirements review status (F14:205). Not all of these practices were prac-

ticed at the studied companies even though some mentioned that such measures would be useful (F14:219).

#### 4.2.10 Job Rotation Practices (P10)

Job rotation was suggested in interviews at companies D and F as a way to improve alignment by extending contact networks and experiences across departments and roles, and thereby supporting spreading and sharing perspectives within an organization. In general, the interviews revealed that alignment is very dependent on individuals, their experience, competence and their ability to communicate and align with others. The practice of job rotation was mentioned as a proposal for the future and not currently implemented at any of the included companies.

### 4.3 Practices that Address the Challenges

This section provides an overview of the relationships between the alignment challenges and practices identified in this study (and reported in Sections 4.1 and 4.2). The mapping is intended as an initial guide for practitioners in identifying practices to consider in addressing the most pressing alignment challenges in their organizations. The connections have been derived through analysis of the parts of the interview transcripts connected to each challenge and practice, summarized in Table 2.6 and elaborated next. The mapping clearly shows that there are many-to-many relations between challenges and practices. There is no single practice that solves each challenge. Consequently, the mapping is aimed at a strategic level of improvement processes within a company, rather than a lower level of practical implementation. After having assessed the challenges and practices of REST alignment within a company, the provided mapping can support strategic decisions concerning which areas to improve. Thereafter, relevant practices can be tailored for use within the specific context. Below we discuss our findings, challenge by challenge.

The practices observed to address the challenge of having *common goals within an organization* (Ch1) mainly concern increasing the synchronization and communication between different units and roles. This can be achieved through involving customers and development-near roles in the requirements process (P1.1, P1.2, P1.3, P1.5); documenting requirement decision rationale (P1.6); validating requirements through test case reviews (P2.1) and product managers reviewing prototypes (P2.3); and involving testing roles in change management (P4.1). Goal alignment is also increased by the practice of basing launch decisions made by management on test reports (P2.4) produced by testers. Furthermore, tracing between artifacts (P6.1–6.4) provides a technical basis for supporting efficient communication of requirements. Job rotation (P10) is mentioned as a long-term practice for

Table 2.6: Mapping of practices to the challenges they are found to address.

|   | P <sub>1</sub> RE practices   | P <sub>2</sub> Validation practices | P <sub>3</sub> Verification practices | P <sub>4</sub> management | P <sub>5</sub> Process enforcement | P <sub>6</sub> between artifacts | P <sub>7</sub> Traceability | P <sub>8</sub> Tool responsibilities | P <sub>9</sub> Alignment metrics | P <sub>10</sub> Job rotation |
|---|-------------------------------|-------------------------------------|---------------------------------------|---------------------------|------------------------------------|----------------------------------|-----------------------------|--------------------------------------|----------------------------------|------------------------------|
| Ch1 Aligning goals and perspectives within organization | P <sub>1.1-1.3, 1.5-1.6</sub> | P <sub>2.1, 2.3-2.4</sub>           | P <sub>3.3</sub>                      | P <sub>4.1</sub>          |                                    | P <sub>6.1-6.4</sub>             |                             |                                      |                                  | P <sub>10 (S)</sub>          |
| Ch2 Cooperating successfully                            | P <sub>1.2-1.3, 1.5-1.6</sub> | P <sub>2.1, 2.3, 2.4</sub>          | P <sub>3.1</sub>                      | P <sub>4.1</sub>          |                                    |                                  |                             |                                      |                                  |                              |
| Ch3 Requirements specification quality                  | P <sub>1.1-1.5</sub>          | P <sub>2.1, 2.5</sub>               |                                       | P <sub>4.1</sub>          | P <sub>5</sub>                     | P <sub>6.2-6.3</sub>             |                             |                                      |                                  | P <sub>9</sub>               |
| Ch4 ST quality  | P <sub>1.1-1.5</sub>          | P <sub>2.1-2.3, 2.5</sub>           | P <sub>3.1-3.3</sub>                  |                           | P <sub>5</sub>                     | P <sub>6.1-6.4</sub>             |                             |                                      |                                  | P <sub>9</sub>               |
| Ch5 Maintaining alignment when requirements change      |                               | P <sub>2.2, 2.5</sub>               |                                       | P <sub>4.1-4.2</sub>      | P <sub>5</sub>                     | P <sub>6.1-6.4</sub>             | P <sub>7</sub>              |                                      |                                  | P <sub>9</sub>               |
| Ch6 Requirements abstraction levels                     |                               | P <sub>1.1, 1.6</sub>               |                                       |                           | P <sub>6.4</sub>                   |                                  |                             |                                      |                                  |                              |
| Ch7 Traceability  | P <sub>2.1</sub>              |                                     |                                       | P <sub>4.1</sub>          | P <sub>5</sub>                     | P <sub>6.1-6.4</sub>             | P <sub>7</sub>              | P <sub>8.1-8.2</sub>                 | P <sub>9</sub>                   |                              |
| Ch8 Time and resource availability                      |                               |                                     |                                       |                           |                                    |                                  |                             |                                      |                                  |                              |
| Ch9 Managing a large document space                     |                               |                                     |                                       |                           |                                    | P <sub>6.1-6.4</sub>             | P <sub>7</sub>              | P <sub>8.1-8.2</sub>                 | P <sub>9</sub>                   |                              |
| Ch10 Outsourcing of components or testing               | P <sub>1.1-1.5</sub>          | P <sub>2.1-2.3</sub>                | P <sub>3.4</sub>                      |                           |                                    | P <sub>6.4</sub>                 |                             |                                      |                                  |                              |

A blank cell indicates that no connection was mentioned during the interviews.

sharing goals and synchronizing perspectives across the organization. In the mid-term perspective, customer feedback received by testers for previous projects (P3.3) can be reused as input when defining road-maps and products plans thereby further coordinating the testers with the requirements engineers responsible for the future requirements.

The challenge of *cooperating successfully* (*Ch2*) is closely related to the first challenge (*Ch1*) as being a means to foster common goals. Practices to achieve close cooperation across roles and organizational borders hence include cross-functional involvement (P1.2, P1.5, P2.4) and reviews (P1.3, P2.1, P2.3), feedback through early and continuous test activities (P3.1), as well as, joint decisions about changes in change control boards (P4.1) and documenting requirement decision rationales (P1.6). The former are practices are embraced in agile processes, while the latter practices of change control boards and documentation of rationales were removed for the studied cases when agile processes were introduced. Job rotation (P10), with its general contribution to building networks, is expected to facilitate closer cooperation across organizational units and between roles.

The challenge of achieving good *requirements specification quality* (*Ch3*) is primarily addressed by the practices for requirements engineering (P1.1–1.5), validation (P2.1, P2.5) and managing requirement changes (P4.1). Some traceability practices (P6.2, P6.3) also address the quality of requirements in terms of being well structured and defined at the right level of detail. Furthermore, awareness of the importance of alignment and full requirements coverage may induce and enable organizations in producing better requirements specifications. This awareness can be encouraged with the use of alignment metrics (P9) or enforced (P5) through regulations for safety-critical software and/or by integrating process adherence in development tools.

The challenge of achieving good *validation and verification quality* (*Ch4*) is addressed by practices to ensure clear and agreed requirements, such as cross-functional reviews (P1.3,P2.1), involving development roles in detailing requirements (P1.2) and customers in defining acceptance criteria (P2.2). Validation is supported by product managers reviewing prototypes (P2.3) and by user/customer testing (P2.5). Verification is improved by early verification activities (P3.1) and through independent testing (P3.2) where testers are not influenced by other engineers' interpretation of the requirements. Complete and up-to-date requirements information is a prerequisite for full test coverage, which can be addressed by requirements engineering practices (P1.1–1.5), testers re-using customer feedback (P3.3) (rather than incorrect requirements specification) and indirectly by traceability practices (P6.1 – 6.4). The external enforcement (P5) of the full test coverage and alignment metrics (P9) are practices that provide incentives for full test coverage including quality requirements.

*Maintaining alignment when requirements change (Ch5)* is a challenge that clearly connects to change and traceability practices (P4.1–4.2, P6.1–6.4 and P7). However, also the validation practices of having acceptance tests based on user scenarios (P2.2) and user/customer testing (P2.5) address this challenge by providing feedback on incorrectly updated requirements, test cases and/or software. Furthermore, having alignment metrics in place (P9) and external regulations on documentation and traceability (P5) is an incentive to maintain alignment as requirements change.

The challenge of managing *requirements abstraction levels (Ch6)* is addressed by the requirements practice of including the customer in requirements work throughout a project (P1.1) and the tracing practices of matching abstractions levels for requirements and test artifacts (P6.4). Both of these practices exercise the different requirements levels and thereby support identifying mismatches. This challenge is also supported by documentation of requirement decision rationales (P1.6) by providing additional requirements information to the roles at the different abstraction level.

*Traceability (Ch7)* in itself is identified as a challenge in the study, and interviewees identified practices on the information items to be traced (P6.1 – 6.4), as well as, tools (P8.1 – 8.2) to enable tracing. In addition, the practice of reviewing test cases against requirements (P2.1) may also support identifying sufficient and/or missing traces. Furthermore, requirements coverage metrics (P9) are proposed as a means to monitor and support traceability. However, as noticed by companies E and F, simple coverage metrics are not sufficient to ensure ample alignment. Process enforcement practices (P5) and assigning specific roles responsible for traceability (P7) are identified as key practices in creating and maintaining traces between artifacts.

The practical aspects of the challenge on *availability of time and resources (Ch8)* are mainly a matter of project management practices, and hence not directly linked to the alignment practices. However, the practice of involving testing roles in the change management process (P4.1) may partly mitigate this challenge by supporting an increased awareness of the verification cost and impact of changes. Furthermore, in companies for which alignment practices are externally enforced (P5) there is an awareness of the importance of alignment of software development, but also an increased willingness to take the cost of alignment including tracing.

The *large document space (Ch9)* is a challenge that can be partly addressed with good tool support (P8.1 – 8.2) and tracing (P6.1 – 6.4, P7) practices. The study specifically identifies that a tool that fits a medium-sized project may be very hard to use in a large one. One way of getting a synthesized view of the level of alignment between large sets of information is to characterize it, using quantitative alignment measurements (P9). It does not solve the large-scale problem, but may help assess the current status and direct management attention to problem areas.

*Outsourcing* (*Ch10*) is a challenge that is related to timing, which is a project management issue, and to communication of the requirements, which are to be developed or tested by an external team. The primary practice to apply to outsourcing is customer communication (P1.1). Frequent and good communication can ensure a common perspective and direction, in particular in the early project phases. In addition, other practices for improved cooperation (P1.2 – P1.5, P2.1 – P2.3) are even more important when working in different organizational units, times zones, and cultural contexts. Furthermore, in an outsourcing situation the requirements specification is a key channel of communication, often also in contractual form. Thus, having requirements and tests specified at the same level of abstraction (P6.4), feasible for the purpose, is a practice to facilitate the outsourcing. Finally, training the outsourced or off-shored team (P3.4) in company practices and tools also addresses this challenge.

In summary, the interviewees brought forward practices, which address some of the identified challenges in aligning requirements and testing. The practices are no quick-fix solutions, but the mapping should be seen as a guideline to recommend areas for long-term improvement, based on empirical observations of industry practice.

## 5 DISCUSSION

Alignment between requirements and test ranges not only the life-cycle of a software development project, but also company goals and strategy, and affects a variety of issues, from human communication to tools and their usage. Practices differ largely between companies of varying size and maturity, domain and product type, etc. One-size alignment practices clearly do not fit all.

A wide collection of alignment challenges and practices have been identified based on the large amount of experiences represented by our 30 interviewees from six different companies, covering multiple roles, domains and situations. Through analyzing this data and deriving results from it, the following general observations have been made by the researchers:

1. The human and organizational sides of software development are at the core of industrial alignment practices.
2. The requirements are the frame of reference for the software to be built, and hence the quality of the requirements is critical for alignment with testing activities.
3. The large difference in size (factor 20) between the companies, in combination with variations in domain and product type, affects the characteristics of the alignment challenges and applicable practices.
4. The incentives for investing in good alignment practices vary between domains.

*Organizational and human issues* are related to several of the identified challenges (Ch1, Ch2, Ch8, and Ch10). Successful cooperation and collaboration (Ch2) is a human issue. Having common goals and perspectives for a development project is initially a matter of clear communication of company strategies and goals, and ultimately dependent on human-to-human communication (Ch1). Failures to meet customer requirements and expectations are often related to misunderstanding and misconception; a human failure although technical limitations, tools, equipment, specifications and so on, also play a role. It does not mean that the human factor should be blamed in every case and for each failure. However, this factor should be taken into account when shaping the work conditions for software engineers. These issues become even more pressing when outsourcing testing. Jones et al. [2009] found that failure to align outsourced testing activities with in-house development resulted in wasted effort, mainly due to weak communication of requirements and changes of them.

Several of the identified alignment practices involve the human and organizational side of software engineering. Examples include communication practices with customers, cross-role and cross-functional meetings in requirements elicitation and reviews, communication of changes, as well as, a proposed job rotation practice to improve human-to-human communication. This confirms previous research that alignment can be improved by increasing the interaction between testers and requirements engineers. For example, including testers early on and, in particular, when defining the requirements, can lead to improved requirements quality [Uusitalo et al., 2008]. However, Uusitalo et al. also found that cross collaboration can be hard to realize due to unavailability of requirements owners and testers on account of other assignments and distributed development [Uusitalo et al., 2008]. In general, processes and roles that support and enforce the necessary communication paths may enhance alignment. For example, Paci et al. [2012] report on a process for handling requirements changes through clearly defined communication interfaces. This process relies on roles propagating change information within their area, rather than relying on more general communication and competence [Paci et al., 2012]. This also confirms the findings of Uusitalo et al. that increased cross communication reduces the amount of assumptions made by testers on requirements interpretation, and results in an increased reliability of test results and subsequent products [Uusitalo et al., 2008]. Similarly, Fogelström and Gorschek [2007] found that involving testers as reviewers through test-case driven inspections of requirements increases the interaction with requirements-related roles and can improve the overall quality of the requirements, thereby supporting alignment. Furthermore, even technical practices, such as tool support for requirements and test management, clearly have a human side concerning degree of usability and usefulness for different groups of stakeholders in an organization.

*Defining requirements of good quality* (Ch3) is central to enabling good alignment and coordination with other development activities, including validation and verification. This challenge is not primarily related to the style of requirements, whether scenario based, plain textual, or formal. But, rather the quality characteristics of the requirements are important, i.e. being verifiable, clear, complete, at a suitable level of abstraction and up-to-date. This relates to results from an empirical study by [Ferguson and Lami \[2006\]](#) that found that unclear requirements have a higher risk of resulting in test failures. A similar reverse relationship is reported by [Graham \[2002\]](#), that clearer and verifiable requirements enable testers to define test cases that match the intended requirements. In addition, [Uusitalo et al. \[2008\]](#) found that poor quality of requirements was a hindrance to maintaining traces from test cases. [Sikora et al. \[2012\]](#) found that requirements reviews is the dominant practice applied to address quality assurance of the requirements for embedded systems and that industry need additional and improved techniques for achieving good requirements quality. Furthermore, requirements quality is related to involving, not only requirements engineers in the requirements engineering, but also ST roles in early stages. This can be achieved by involving non-RE roles in reviews and in detailing requirements. This also contributes to cross-organizational communication and learning, and supports producing requirements that are both useful and used. [Uusitalo et al. \[2008\]](#) found that testers have a different viewpoint that makes them well suited to identifying deficiencies in the requirements including unverifiability and omissions. [Martin and Melnik \[2008\]](#) take this approach one step further by suggesting that the requirements themselves be specified as acceptance test cases, which are then used to verify the behavior of the software. This approach was evaluated through an experiment by [Ricca et al. \[2009\]](#) who found that this helped to clarify and increase the joint understanding of requirements with substantially the same amount of effort. Furthermore, our findings that RE practices play a vital role in supporting REST alignment confirm previous conclusions that the requirements process is an important enabler for testing activities and that RE improvements can support alignment with testing [[Uusitalo et al., 2008](#)].

*Company size* varies largely between the six companies in this study. Similarly, the challenges and practices also vary between the companies. While smaller project groups of 5–10 persons can handle alignment through a combination of informal and formal project meetings. Large-scale projects require more powerful process and tool support to ensure coordination and navigation of the (larger) information space between different phases and hierarchies in the organization. This was illustrated by different views on the same state-of-the-art requirements management tool. The tool supported alignment well in one medium-sized project (company C), but was frequently mentioned by the interviewees for the largest company (com-

### Incentives for alignment practices

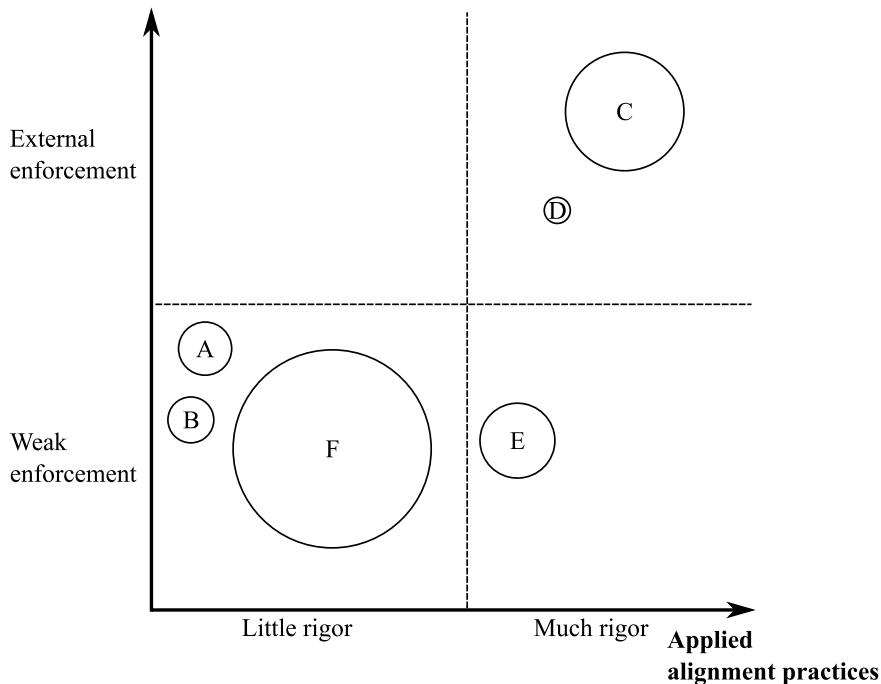


Figure 2.4: Rough overview of the relationship between the variation factors size, rigor in applying alignment practices and incentive for alignment practices for the studied companies. Number of people in software development is reflected by the relative size of the circle.

pany F) as a huge alignment challenge. In some cases (e.g. company F), agile practices are introduced to manage large projects by creating several smaller, self-governing and less dependent units. Our study shows that this supports control and alignment at the local level, but, at the expense of global control and alignment (company E). The size-related alignment challenges then re-appear in a new shape, at another level in the organization. For example, granting development teams mandate to define and change detailed requirements increases speed and efficiency at the team level, but increases the challenge of communicating and coordinating these changes wider within a large organization.

*The incentives for applying alignment practices*, specifically tracing between requirements and test artifacts, vary across the studied companies. Applying alignment practices seems to be connected to the incentives for enforcing certain practices, such as tracing and extensive documentation. The

companies reporting the most rigid and continuously maintained alignment practices are those working in domains where customers or regulatory bodies require such practices. Both of these companies (C and D) have enforced alignment practices in their development including tracing between requirements and tests. Interestingly these are also the companies in our study which apply a traditional and rigorous development model. It is our interpretation that the companies with the most agile, and least rigorous, development processes (A and B) are also the companies which rely heavily on people-based alignment and tracing, rather than on investing in more structured practices. These are also the two companies that do not have tracing between artifacts in place, even partially. While for the remaining companies (E and F) which apply agile-inspired processes, but with structured elements (e.g. eRUP), traceability is in place partly or locally. Our interpretation of the relationship between the included companies concerning incentives and degree of rigor in applying structured alignment practices is illustrated in Figure 2.4 together with the relative size of their software development. The observed connection between degree of rigor and incentives for alignment are similar to other findings concerning safety-critical development. Namely, that alignment is enabled by more rigorous practices such as concurrently designed processes [Kukkanen et al., 2009] or model-based testing [Nebut et al., 2006, Hasling et al., 2008]. Furthermore, companies in safety-critical domains have been found to apply more rigorous processes and testing practices [Runeson et al., 2003]. In contrast, neglect of quality requirements, including safety aspects has been found to one of the challenges of agile RE [Cao and Ramesh, 2008].

Interestingly, several alignment challenges (e.g. tracing, communicating requirements changes) were experienced also for the companies developing safety-critical software (C and D) despite having tracing in place and applying practices to mitigate alignment challenges (e.g. frequent customer communication, tracing responsible role, change management process involving testers etc.) This might be explained by a greater awareness of the issues at hand, but also that the increased demands posed by the higher levels of quality demands requires additional alignment practice beyond those needed for non-critical software.

When the documentation and tracing practices are directly enforced from outside the organization, they cannot be negotiated and the cost has to be taken [Watkins and Neal, 1994]. In organizations without these external requirements the business case for investing in these practices needs to be defined, which does not seem to be the case for the studied organizations. Despite the existence of frustration and rework due to bad alignment, the corresponding costs are seldom quantified at any level. Improving alignment involves short term investments in tools, work to recapture traces between large legacies of artifacts, and/or in changed working prac-

tices. The returns on these investments are gained mainly in the longer term. This makes it hard to put focus and priority on alignment practices in a shortsighted financial and management culture. Finally, requirements volatility increases the importance and cost to achieve REST alignment. This need to manage a rate of requirements changes often drives the introduction of agile practices. These practices are strong in team cooperation, but weak in documentation and traceability between artifacts. The companies (C and D) with lower requirements volatility and where development is mainly plan-driven and bespoke, have the most elaborated documentation and traceability practices. In both cases, the practices are enforced by regulatory requirements. However, in our study, it is not possible to distinguish between the effects of different rates of change and the effects of operating in a safety-critical domain with regulations on documentation and traceability.

In summary, challenges and practices for REST alignment span the whole development life cycle. Alignment involves the human and organizational side of software engineering and requires the requirements to be of good quality. In addition, the incentives for alignment greatly vary between companies of different size and application domain. Future research and practice should consider these variations in identifying suitable practices for REST alignment, tailored to different domains and organizations.

## 6 CONCLUSIONS

Successful and efficient software development, in particular on the large scale, requires coordination of the people, activities and artifacts involved [Kraut and Streeter, 1995, Damian et al., 2005, Damian and Chisan, 2006]. This includes alignment of the areas of requirements and test [Damian and Chisan, 2006, Uusitalo et al., 2008, Kukkanen et al., 2009, Sabaliauskaitė et al., 2010]. Methods and techniques for linking artifacts abound including tracing and use of model-based engineering. However, companies experience challenges in achieving alignment including full traceability. These challenges are faced also by companies with strong incentives for investing in REST alignment such as for safety critical software where documentation and tracing is regulated. This indicates that the underlying issues lie elsewhere and require aligning of not only the artifacts, but also of other factors. In order to gain a deeper understanding of the industrial challenges and practices for aligning RE with ST, we launched a case study covering six companies of varying size, domain, and history. This chapter reports the outcome of that study and provides a description of the industrial state of practice in six companies. We provide categorized lists of (RQ1) industrial alignment challenges and (RQ2) industrial practices for improving alignment, and (RQ3) a mapping between challenges and

practices. Our results, based on 30 interviews with different roles in the six companies, add extensive empirical input to the existing scarce knowledge of industrial practice in this field [Uusitalo et al., 2008, Sabaliauskaitė et al., 2010]. In addition, this chapter presents new insights into factors that explain needs and define solutions for overcoming REST alignment challenges.

We conclude with four high-level observations on the alignment between requirements and testing. Firstly, as in many other branches of software engineering, the human side is central, and communication and coordination between people is vital, so also between requirements engineers and testers, as one interviewee said: “start talking to each other!” (F7:88). Further, the quality and accuracy of the requirements is a crucial starting point for testing the produced software in-line with the defined and agreed requirements. Additionally, the size of the development organization and its projects is a key variation factor for both challenges and practices of alignment. Tools and practices may not be scalable, but rather need to be selected and tailored to suit the specific company, size and domain. Finally, alignment practices such as good requirements documentation and tracing seem to be applied for safety-critical development through external enforcement. In contrast, for non-safety critical cases only internal motivation exists for the alignment practices even though these companies report facing large challenges caused by misalignment such as incorrectly implemented requirements, delays and wasted effort. For these cases, support for assessing the cost and benefits of REST alignment could provide a means for organizations to increase the awareness of the importance of alignment and also tailor their processes to a certain level of alignment, suitable and cost effective for their specific situation and domain.

In summary, our study reports on the current practice in several industrial domains. Practical means are provided for recognizing challenges and problems in this field and matching them with potential improvement practices. Furthermore, the identified challenges pose a wide assortment of issues for researchers to address in order to improve REST alignment practice, and ultimately the software engineering practices.

# 3

---

## A TAXONOMY FOR REQUIREMENTS ENGINEERING AND SOFTWARE TEST ALIGNMENT

---

**SUMMARY** Requirements Engineering and Software Testing are mature areas and have seen a lot of research. Nevertheless, their interactions have been sparsely explored beyond the concept of traceability. To fill this gap we propose a definition of requirements engineering and software test (REST) alignment, a taxonomy that characterizes the methods linking the respective areas, and a process to assess alignment. The taxonomy can support researchers to identify new opportunities for investigation, as well as practitioners to compare alignment methods and evaluate alignment, or lack thereof. We constructed the REST taxonomy by analyzing alignment methods published in literature, iteratively validating the emerging dimensions. The resulting concept of an information dyad characterizes the exchange of information required for any alignment to take place. We demonstrate use of the taxonomy by applying it on five in-depth cases and illustrate angles of analysis on a set of thirteen alignment methods. In addition we developed an assessment framework (REST-bench), applied it in an industrial assessment, and showed that it, with a low effort, can identify opportunities to improve REST alignment. Although we expect that the taxonomy can be further refined, we believe that the information dyad is a valid and useful construct to understand alignment.

### 1 INTRODUCTION

Industrial-scale software development is an undertaking that requires judicious planning and coordination of the involved resources. The inception, design, implementation, examination and maintenance of a software product [Scacchi, 2001] are a team effort, organized and executed to satisfy the product customer. Following the separation of concerns principle, software life-cycle models distinguish between different phases or activities in the production of software, linking them by feed-forward and feed-back loops [Madhavji, 1991]. This separation reduces the complexity of each single

phase or activity, however at the same time poses needs for an efficient and effective coordination.

We investigate two phases in the software development life-cycle, requirements engineering (RE) and software testing (ST), that benefit particularly from a coordinated functioning [Graham, 2002]. Several prominent researchers have called for more research towards this goal. At FoSE 2007, Cheng and Atlee [2007] called for a stronger collaboration between RE and researchers and practitioners from other software engineering fields to improve requirements knowledge and downstream development. Bertolino [2007] summarized current challenges and goals in software testing research, pointing out the rising importance of a more holistic approach to ST which takes advantage of the overlaps between different research disciplines. Recent research shows that the study of the synergies between RE and ST are important and of particular interest for industry [Uusitalo et al., 2008, Post et al., 2009, Sabaliauskaitė et al., 2010].

Despite these advancements and its relevance for practitioners, there is still a lack of research that aims at understanding, characterizing and communicating methods that align requirements engineering and software test. By studying methods for RE and ST alignment we intend to fill this gap. This chapter does not aim at providing a systematic and exhaustive state-of-the-art survey of RE or ST research, but rather forms the foundation, through a taxonomy, to classify and characterize alignment research and solutions that focus on the boundary between RE and ST. The REST taxonomy also functions as an engine for REST-bench, an alignment assessment framework.

With *alignment* we mean the *adjustment of RE and ST efforts for coordinated functioning and optimized product development*. Depending on the context, alignment can be understood as an activity or as a state. Alignment-as-activity pertains to the act of *adjusting or arranging* efforts involved in RE and ST so that they work better together. To improve our understanding of such activities, we developed the REST taxonomy. Alignment-as-state, on the other hand, refers to the condition of RE and ST efforts having *established* a coordinated functioning. In order to evaluate the state of alignment we developed REST-bench which acts as an assessment framework and is based on the REST taxonomy. Independently from the context, the above definitions imply that a higher degree of alignment enables higher effectiveness and efficiency in product development and/or maintenance.

We study RE and ST alignment with the purpose of

- *Characterization* of RE and ST alignment methods, providing practitioners and researchers a common vocabulary
- *Analysis* of RE and ST alignment methods, providing researchers means to preemptively identify weaknesses and suggest improvements

- *Industrial assessment* of RE and ST alignment, providing practitioners a lightweight framework (REST-bench, powered by the REST taxonomy) to identify misalignment

The remainder of the chapter is structured as follows. In Section 2 we discuss the relationship between requirements engineering and software testing in more detail and illustrate related work. In Section 3 we present the REST taxonomy, accompanied with an example of its application, and classify thirteen alignment methods. In Section 4 we illustrate the process followed for constructing and validating the taxonomy. In Section 5 we analyze the classified methods by the means the REST taxonomy provides. We introduce REST-bench, which we applied in an industrial case study at Ericsson AB, in Section 6. The chapter concludes with Section 7, pointing out directions for future work.

## 2 BACKGROUND AND RELATED WORK

### 2.1 The Need for Alignment

Software development consists of transitions from system concept, requirements specification, analysis and design, implementation, and test and maintenance [Laplante, 2007]. This abstraction holds for both plan driven process models (e. g. spiral [Boehm, 1988] and evolutionary [Naumann and Jenkins, 1982], and the unified process model [Kruchten, 2000]), as well as and Agile models, although to a lesser extent in the latter category as activities may be blended, eliminating transitions altogether (e. g. in eXtreme Programming [Beck, 1999]).

Looking at the V-Model, which originates from system engineering [Forsberg and Mooz, 1991, Bröhl and Dröscher, 1995] and was adopted in software engineering [Pfleeger and Atlee, 2009], high-level testing is often depicted as the Verification & Validation activity to requirements elicitation, analysis and specification. As such, this connection between requirements engineering and testing is a key part of our software engineering knowledge. Still, this connection is not considered in detail as a collective concept in our research activities. On the other hand, an abundance of software technologies, models and frameworks have been developed to ease the transition of software development phases, to bridge the gap between them, and to *align* the intentions and activities therein, for example, between requirements and software architecture/design ([Kop and Mayr, 1998, Amyot and Mussbacher, 2001, Hall et al., 2002]), software architecture/design and implementation ([Murphy et al., 2001, Elrad et al., 2002, Aldrich et al., 2002]), and software architecture/design and testing ([Muccini et al., 2004, Samuel et al., 2007]).

However, aligning requirements engineering and software testing is a less explored territory, although it would be beneficial to recognize the inherent link between them [Graham, 2002]. The need for RE and ST alignment is emphasized by the difficulty to design, implement and maintain large software systems. The increase in complexity of the problem space, i.e. requirements, increases also the complexity of the software solution [Glass, 2002b], making therefore the testing more involved. Benefits of a strengthened link between RE and ST are, for example, improved product quality [Uusitalo et al., 2008], cost-effective testing [Miller and Strooper, 2010, Flammini et al., 2009], high quality test-cases [de Santiago Júnior and Vijaykumar, 2012], and early discovery of incomplete requirements [Siegl et al., 2010].

The means by which RE and ST alignment can be achieved, include (but are not limited to) methods or processes that establish and maintain requirements to test traceability links [Gotel and Finkelstein, 1994, Ramesh and Jarke, 2001], use requirements as a driver to develop tests (e.g. by formulating testable contracts [Melnik et al., 2006, Martin and Melnik, 2008], use model-based testing [Utting et al., 2012]), or organize development teams in an effective manner (e.g. by forming cross-functional teams [Marczak and Damian, 2011]).

The means of achieving alignment are diverse in terms of the assumptions they make, their prerequisites on the organizational environment, and the investments they require. Effectively searching, selecting and applying instruments to improve RE and ST alignment is therefore a challenge for practitioners but also for researchers in advancing the state-of-the-art. Uusitalo et al. [2008] conducted interviews at five Finnish software organizations and elicited practices, such as tester participation in requirements reviews, and requirements to test traceability, that aim to bridge the gap between RE and ST. Post et al. [2009] explored how the impact of requirements changes, and the subsequent effort in adapting test cases, can be reduced by scenario-based requirements formalizations. In an interview study with software practitioners occupying roles as quality control leaders, requirements process managers and test leaders, Sabaliauskaitė et al. identified several obstacles in aligning requirements engineering and testing. Barriers exist in the organizational structure, processes and cooperation between people, and are aggravated by tool deficiencies and challenges in change management [Sabaliauskaitė et al., 2010].

The connections between RE and ST are both clear and numerous, and the potential benefits in increasing the coordination between them are large. Therefore it is essential that we increase our understanding through the study of these connections, and treat them as a collective and not as individual, isolated areas and approaches. Our main aim in this chapter is to systematically create a basis for such an understanding. In order to char-

acterize the phenomenon of alignment between RE and ST we developed therefore the REST taxonomy.

## 2.2 *Alignment vs. Traceability*

The concept of traceability, which exists since the dawn of the software engineering discipline [Randell, 1968], is not associated with a particular goal, but is a quality attribute of the artifacts produced in software development. The IEEE Standard Glossary of Software Engineering Terminology [IEEE, 1990] defines traceability as “the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [...]”. Gotel and Finkelstein provide a similar definition of requirements traceability as “the ability to describe and follow the life of a requirement” [Gotel and Finkelstein, 1994], which complies with the notion of traceability being a work product quality attribute.

Research into traceability indicates that good traceability supports impact analysis [Gotel and Finkelstein, 1994, Ramesh and Jarke, 2001, Damian et al., 2005, Uusitalo et al., 2008] and lowers test and maintenance costs [Watkins and Neal, 1994, Kukkanen et al., 2009]. On the other hand, high quality traces are expensive to establish and maintain [Cleland-Huang et al., 2003], leading to the investigation of means to automate the trace recovery process [De Lucia et al., 2007, Hayes et al., 2007].

We defined alignment as a goal-directed concept, i.e. the adjustment of RE and ST efforts for coordinated functioning and optimized product development. As such, high quality traces may contribute to an improved alignment, are however not the only solution candidates achieving our goal of alignment. Thus, traceability can be a method to achieve alignment, but the REST taxonomy focuses on the alignment phenomena itself and how methods for alignment (which might build on traceability) can be classified.

## 2.3 *The Purpose of Taxonomies*

Creating taxonomies of objects or concepts has been a basic scientific tool since early work by the Swedish botanist Carl von Linné [Linnaei, 1735]. Taxonomies are means to structure, advance the understanding, and to communicate knowledge [Glass and Vessey, 1995, Kwasnik, 1999]. When the understanding in a certain area advances, concepts and relationships between them emerge that allow for a structured representation of these concepts. Being able to communicate that knowledge provides the opportunity to further advance research [Kwasnik, 1999]. Kwasnik also points out the importance of taxonomies as theory developing tools. Classification

schemes enable the display of theory in an useful way and serve, similar to theories, as drivers for inquiry [Kwasnik, 1992]. Thus, the development of taxonomies is essential to document theories which accumulate knowledge on Software Engineering phenomena [Sjøberg et al., 2007].

## 2.4 *Taxonomies in Software Engineering*

The Guide to the Software Engineering Body of Knowledge (SWEBOK) is an attempt to characterize the software engineering discipline and to provide a structured access to its body of knowledge [Bourque and Dupuis, 2004]. As such, SWEBOK can be seen as a taxonomy that covers knowledge areas relevant to software engineering, promoting the structured communication of this discipline. Similarly, Glass et al. [2002] provide a taxonomy on the research in software engineering, although its main purpose is to structure and to position past research. Blum's taxonomy of software development methods [Blum, 1994] is more narrow in scope and, similar to Glass et al., aims at structuring rather than communicating the knowledge on software development methods.

Further examples of specialized taxonomies, i. e. with a narrow scope, are Buckley et al. [2005] on mechanisms of software change, Svahnberg et al. [2005] on variability realization techniques, and Mehta et al. [2000] on software component connectors.

## 2.5 *Developing Taxonomies*

The development of a taxonomy can be approached in two different ways, top-down and bottom-up [Glass et al., 2002]. In the top-down or enumerative [Broughton, 2004] approach, the classification scheme is defined a priori, i. e. a specific structure and categories are established that aim to fulfill the purpose of the taxonomy. The created classification scheme is thereby often a composition of previously established schemata (e. g. Glass et al. [2002], Avižienis et al. [2004], Bunse et al. [2006]), or the result of the conceptual analysis of a certain area of interest (e. g. Svahnberg et al. [2005], Utting et al. [2012]). The strength of this approach is that the taxonomy is built upon existing knowledge structures, allowing the reuse of established definitions and categorizations and hence increasing the probability of achieving an objective classification procedure.

On the other hand, the bottom-up or analytico-synthetic approach [Broughton, 2004] is driven by the sampling of subjects from the population of interest and the extraction of patterns that are refined into a classification scheme. For example, Vegas et al. [2009] extended existing unit-testing classifications by systematically studying the Software Engineering literature, supplemented by gathering the expert judgment of researchers

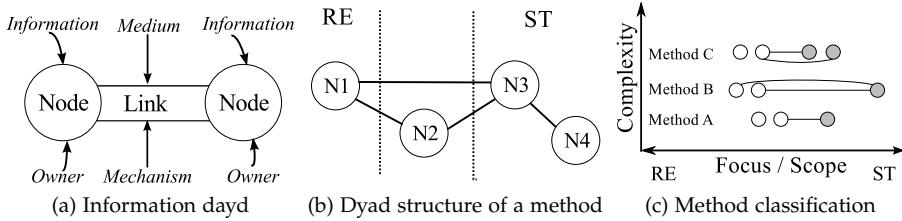


Figure 3.1: Anatomy of the REST taxonomy

and practitioners in the testing area. The strength of this approach is that new, not yet classified, characteristics may emerge and enrich existing taxonomies.

The goal of the taxonomy presented in this chapter is to classify methods that bridge the gap between requirements engineering and software testing activities. There exists a rich knowledge base for both RE and ST, and taxonomies for classifying aspects in each area already exist. Following a top-down approach and amalgamating concepts, definitions and categorizations from these separate areas into a taxonomy of RE and ST alignment seemed to us unlikely to succeed. Even though the respective areas are mature and have seen a lot of research, their interplay and connections have been less explored. Hence we chose to construct the taxonomy in a bottom-up fashion, validating the emerging classification scheme throughout the process (see Section 4).

### 3 THE REST TAXONOMY

When developing a taxonomy one has to consider its *purpose* [Glass and Vessey, 1995]. A specific taxonomy is designed to accommodate a single, well-defined purpose. On the other hand, the structure of general taxonomy is not imposed by a specific purpose [Glass and Vessey, 1995] and is hence applicable in various circumstances. As we defined earlier in Section 1, alignment can be understood as an activity or a state. We therefore designed the structure of our taxonomy to accommodate both aspects of the alignment definition. From the alignment-as-activity perspective, the REST taxonomy can be used to analyze and categorize alignment methods described in literature. From the alignment-as-state perspective, the REST taxonomy serves as an analysis aid in project and process assessment. The method we developed for process assessment (which connects to alignment-as-state), REST-bench, is described and illustrated through a case study in Section 6.

Figure 3.1 provides an overview of the REST taxonomy. The taxonomy is centered around our observation (see Section 4.1) that the alignment of

RE and ST implies some sort of information linkage or transfer between two entities involved in either process area. In essence, if there is no exchange of information, at least at some point in time, no alignment can take place or be achieved. Thus, characterizing such exchanges are key in a general taxonomy. In order to describe this phenomenon we devised the concept of an information dyad, representing the central unit of analysis in the taxonomy. The information dyad contains the *criteria for differentiation and description* [Glass and Vessey, 1995] used for the classification, the second essential aspect in taxonomy development. Figure 3.1a illustrates the components of an information dyad. A *node* is characterized by the type of information it represents and an owner of that information. Two nodes are connected by a *link*, characterized by the type of mechanism establishing the link between nodes, and the medium through which the link is realized.

The third important property of a taxonomy, besides its purpose and criteria for differentiation, is the *method of classification* [Glass and Vessey, 1995]. The method should illustrate and explain how objects under study are classified in a repeatable and unambiguous manner. To this end we developed a process, summarized in Table 3.1, in which each step answers a specific question.

The objects under study are methods that may improve the alignment between RE and ST, published at conferences or in journals. Hence, Step 1 in the process serves as a gatekeeper, asserting that the taxonomy is applied on studies that can answer the questions asked in the following steps. Steps 2.1, 2.2 and 2.3 aim at identifying the information dyads of the studied method, and characterizing the dyads by their components (information, medium and mechanism). The context in which the alignment method has been developed or applied is captured in Step 3.

Since an alignment method consists of one or more dyads, these dyads form a structure which characterizes the method (Figure 3.1b). In Step 4 we analyze the properties of the dyad structure which allows us in Step 5 to classify the methods according to their complexity and scope/focus (Figure 3.1c).

The remainder of this section describes the process shown in Table 3.1. To complement the description, we illustrate the application of the taxonomy by self-contained examples, based on Miller and Strooper [2010] case study on their framework and method of model-based testing of specifications and implementations. Note that section and figure numbers in the examples refer to Miller and Strooper [2010]. Finally, we apply the taxonomy on 13 alignment methods in Section 3.5.

Table 3.1: REST classification process

| Step | Question answered                                     | Section                                       |
|------|---|---|
| 1    | Does the study shed light on both RE and ST aspects?  | <a href="#">3.1 Relevance</a>                 |
| 2    | What are the components of the information dyad?      | <a href="#">3.2 The information dyad</a>      |
| 2.1  | What type of information exists/is used in RE and ST? | <a href="#">3.2.1 Information</a>             |
| 2.2  | What type of medium connects the information?         | <a href="#">3.2.2 Medium</a>                  |
| 2.3  | What type of mechanism establishes the connection?    | <a href="#">3.2.3 Mechanism</a>               |
| 3    | In which environment is the method situated?          | <a href="#">3.3 Method context</a>            |
| 4    | What is the structure of the identified dyads?        | <a href="#">3.4 Dyad structure properties</a> |
| 5    | How can the method be classified?                     | <a href="#">3.5 Method classification</a>     |

### 3.1 Relevance

The analyst (the person who applies the taxonomy) needs to decide whether the study, and the described alignment method therein, qualifies to be classified with the taxonomy. He bases his decision on three independent criteria:

- *Scope*: Since the taxonomy aims at characterizing links between requirements engineering and software testing activities, the candidate study should consider both areas in the discussion of the presented method. If, for example, the focus of the study is on formal reviews of requirement specifications, considers however also the effects of reviews on downstream development including testing or discusses the involvement of quality assurance personnel in reviews, the study is likely to be adequate for taxonomy application. On the other hand, a comparison of different review techniques, focused on identifying respective strengths and weaknesses alone, is likely not to be adequate.
- *Comprehensiveness*: A detailed report of the conducted study reduces the analyst's leeway for interpretation when answering the questions posed in Table 3.1. It is impossible to judge the comprehensiveness of a publication a-priori, i.e. before reading it, but since space restrictions of journals are less rigid than for conference or workshop publications, they tend to exhibit more details on the conducted study.
- *Rigor*: In case the publication includes a method evaluation, rigor of reporting context, design and validity threats [Ivarsson and Gorschek, 2011] should be considered. A strong description of these as-

Example: Step 1 – Does the study shed light on both RE and ST aspects?

[Miller and Strooper, 2010] present and evaluate a framework that aims at taking advantage of synergy effects between specification testing and implementation testing. They argue that the effort spent in verifying the formal specification of a software system can also contribute to the verification of the implementation of that system. To this end, they introduce testgraphs as a mean to model parts of the specification that require testing, using them for the derivation of test sequences in both specification and implementation testing.

The table below illustrates our assessment with respect to the relevance criteria we defined.

| Criterion         | Strength   | Weakness  |
|-------------------|--|---|
| Scope             | <ul style="list-style-type: none"> <li>+ includes both RE and ST activities (Section 3)</li> <li>+ consistency and conformance check between testgraph and formal specifications (Section 3.1)</li> </ul>      | <ul style="list-style-type: none"> <li>- derivation of formal specifications from requirements is not part of the framework (Figure 2)</li> </ul> |
| Comprehensiveness | <ul style="list-style-type: none"> <li>+ activities respectively roles are described (Section 3, Section 5)</li> </ul>   |   |
| Rigor             | <ul style="list-style-type: none"> <li>+ context (Sections 4.2-4.4) and design (Section 4.1, Section 7) described</li> <li>+ risks in the application of the framework are considered (Section 3.3)</li> </ul> | <ul style="list-style-type: none"> <li>- threats to the validity of the evaluation not discussed</li> </ul>                                       |

Based on this assessment, we conclude that we can apply the REST taxonomy on the described method.

pects supports the analyst in performing context identification (Step 3, Section 3.3).

The example for Step 1 shortly introduces the publication on which all following examples in this section are based upon, and illustrates the application of the above discussed criteria to assess relevance.

### 3.2 The Information Dyad

The goal of this step is to identify the nodes and the link that characterize an information dyad (see Figure 3.1a). Note however that an alignment method can be described by more than one dyad, depending on the number of identified nodes. Hence we discuss dyad structures and their properties in Section 3.4.

#### 3.2.1 Information

An information dyad consists of two nodes and a connecting link. A node describes an entity that has to be aligned, synchronized, brought into agreement, with another entity. The nodes represent the different, primary objects of information, while the link represents the fact that they are or

should affect one or both of each other. To differentiate between nodes, we assign each node a name, characterizing its purpose. We deliberately do not limit the definition of a node to the notion of, for example, a phase in the software development life-cycle. A node could also be an activity, e.g. formal inspections *during* requirements analysis. Although this allows for more flexibility, it also reduces the repeatability in the classification of the alignment method.

A node is characterized by the information it contains and an owner who is the source of that information. In this work, we informally define information as a *coherent collection of related data that is created during software development, often with a specific purpose in mind*. Later on in Section 3.2.3 we will further refine the notion of the information concept, but for this step in the classification process this operational definition is sufficient.

Information, according to the above definition, is created, recorded and used at any point in time during software development, enabling product inception, specification, implementation, verification and validation, and maintenance. Typically it refers to development artifacts but it can also represent more intangible but essential and purposely related knowledge of a developers or testers [Feldt, 2002], e.g. informal requirements as in the Miller and Strooper case (see the example for Step 2.1). With this taxonomy we aim to capture in particular information that is shared and aligned in RE and ST activities. This does however not restrict the information content to RE or ST topics, e.g. technical requirements, feature descriptions, priorities, test plans, strategies, scenarios, etc. Information valuable for alignment can emerge from any phase in software development and connect RE and ST activities. Hence, the task of the analyst is to carefully study the described method and collect evidence for the existence of a node and its characterizing information. Such evidence can be found in statements on used or created artifacts or in descriptions of things that have been discussed or communicated. The owner, the second attribute of a node, is responsible for creating and/or maintaining the information. Depending on the organization of the development process, the owner may be formally assigned to this responsibility (i.e. by occupying a specific role or function) or, in case of agile processes, depend on the employee's current activities.

### 3.2.2 Medium

The Oxford English Dictionary defines the term medium as “an intermediate agency, instrument, or channel; a means; especially a means or channel of communication or expression” [Oxford English Dictionary, 2011]. The medium in an information dyad describes how the information between two nodes is linked together. This can be through a carrier of information, e.g. an artifact, or a facilitator that enables the information transfer, e.g. a

### Example: Step 2.1 – What type of information exists/is used in RE and ST?

To answer this question, we focus first on identifying actors and the information on which is acted upon. The framework description and the illustration of the performed tasks in the GSM case study are thereby of interest (Section 3, 3.1, 3.2, 5.1, 6.1 and 6.2). Remember that our ultimate goal in this step is to identify potential nodes that form one or more information dyads. Hence the other components, medium and mechanism, play a secondary role at this moment. Defining the characteristics of links too early in the process may inhibit the discovery of all relevant nodes. On the other hand, since Steps 2.1-2.3 are performed iteratively, refinements are still possible at a later moment. The table below lists the identified nodes. In the following we will motivate them and illustrate in which relation they stand to each other, i. e. define the information dyads in this example.

| ID             | Node name             | Information                | Owner     |
|----------------|-----------------------|----------------------------|-----------|
| N <sub>1</sub> | Req. specification    | Informal requirements      | Req. eng. |
| N <sub>2</sub> | Req. analysis (Impl.) | Formal specification       | Req. eng. |
| N <sub>3</sub> | Req. analysis (Test)  | Testgraphs                 | Tester    |
| N <sub>4</sub> | Specification test    | Test sequence              | Tester    |
| N <sub>5</sub> | Specification mapping | Spec. to impl. mapping     | Tester    |
| N <sub>6</sub> | Testgraph mapping     | Testgraph to impl. mapping | Tester    |
| N <sub>7</sub> | Implementation test   | Test sequence              | Tester    |

Dyads (6): N<sub>1</sub>-N<sub>2</sub>, N<sub>1</sub>-N<sub>3</sub>, N<sub>3</sub>-N<sub>4</sub>, N<sub>2</sub>-N<sub>5</sub>, N<sub>3</sub>-N<sub>6</sub>, N<sub>6</sub>-N<sub>7</sub>

The first node, N<sub>1</sub>, contains the fundamental information, i. e. informal requirements, from which further artifacts are derived. The formal specification is developed by the requirements engineer to aid design and implementation. Hence we define requirements analysis (Implementation) as the second node (N<sub>2</sub>). Similarly, the tester develops a testgraph, with associated test cases, to aid the verification of the formal specification and the implementation. Requirements analysis (Test) is therefore the third node (N<sub>3</sub>). The dyads, N<sub>1</sub>-N<sub>2</sub> and N<sub>1</sub>-N<sub>3</sub>, follow from the refinement performed by requirements engineers and testers. The specification is tested by generating test sequences (N<sub>4</sub>) from the testgraph, leading to dyad N<sub>3</sub>-N<sub>4</sub>. Identifying the next nodes is rather challenging. Figure 2 identifies the test oracle and the implementation as further artifacts relevant for the framework. From the alignment perspective however, the interesting part of the framework is the mapping between specification respectively testgraph to the implementation, described in Sections 6.1 and 6.2. The reason why it is interesting is that the tester needs to process and understand artifacts developed by requirements engineers (formal specification) and developers (implementation). Hence, specification mapping (N<sub>5</sub>) and testgraph mapping (N<sub>6</sub>) are nodes of interest, leading to dyads N<sub>2</sub>-N<sub>5</sub> and N<sub>3</sub>-N<sub>6</sub> representing the relationships of the mapping. The implementation test (N<sub>7</sub>), also based on the derivation of a test sequence, is driven by the testgraph mapping, leading to dyad N<sub>6</sub>-N<sub>7</sub>.

### Example: Step 2.2 – What type of medium connects the information?

For each dyad identified in Step 2.1 we now define their linking medium.

In the dyad Requirements specification – Requirements analysis (Implementation), a requirements engineer is responsible for deriving the formal specification from informal requirements. We have to assume that this derivation is performed manually, following a certain process, since the framework description does not explain this step in detail. Hence, for the dyad N1-N2 we declare the medium to be a process. Similarly, testgraphs are derived by a tester from the informal requirements, represented by the dyad Requirements specification – Requirements analysis (Test). Also here, the derivation is a series of activities (define testgraph and associated test cases, measure specification coverage) that follows standard testing heuristics. Hence we declare also in the dyad N1-N3 the medium to be a process. The generation of test sequences (N4) is supported by a tool for the editing the graph and executing tests, leading to the conclusion that the medium in dyad N3-N4 is a tool. Both in dyad N2-N5 and N3-N6, in which mappings between a model (specification respectively testgraphs) and the implementation are created, the link medium is a process. The tester implements wrapper classes for the classes under test (dyad N2-N5), linking state, operations, input and output, and return values from the implementation to the corresponding entities in the specification. In dyad N3-N6, the tester performs a similar task by implementing a driver class that calls for each traversed node and arc in the testgraph the appropriate operation in the wrapper class. Although both mappings can be potentially created automatically, such a tool is currently not available in the framework. On the other hand, the generation of test sequences for the implementation test (N7), is tool supported. Hence the medium in dyad N6-N7 is a tool.

process. During the development of the taxonomy we have identified a set of different media types:

- Structured artifacts (e.g. documents, email, diagrams, database records); they are usually persistent and searchable/indexed.
- Unstructured artifacts (audio, video); they are usually not searchable/indexed.
- Tools that act as means to share, transfer or transform information (e.g. modeling tools, language analysis tools).
- Process (one or more activities, can be performed repeatedly).
- Organization of work environment (co-location, role/responsibility rotation).

The analyst can choose one of these media types if appropriate or introduce a new type as the above set was derived only from a sample of alignment methods studied and hence be incomplete.

#### 3.2.3 Mechanism

The mechanism component of a dyad link characterizes the way in which information is carried, eventually changing its purpose, from one node to the next. We assume that a node in a dyad fulfills a certain purpose in the development of software and is hence embedded in a context that supports the realization of that purpose. For example, requirements analysis is performed at a certain point in time by people possessing the knowledge

### Example: Step 2.3 – What type of mechanism establishes the connection?

We start by looking at the dyads that contain both N<sub>1</sub>, informal requirements, as an information characteristic in the node. The information in both N<sub>2</sub> and N<sub>3</sub> is derived, although by different roles, from the informal requirements. The mechanism for this derivation is in both cases not explicitly specified in the framework. Hence the connection between the nodes is in both dyads an *implicit* one. The mechanism in dyad N<sub>3</sub>-N<sub>4</sub> is however a *transformation* as test sequences are extracted from the testgraph which are used to animate and test the specification.

Dyads N<sub>2</sub>-N<sub>5</sub> and N<sub>3</sub>-N<sub>6</sub>, on the other hand, are explicitly connected by the tester, creating a mapping between the implementation and the specification respectively the testgraph. The mere mapping between information in these dyads does not fulfill the requirements of a transformation mechanism. Consider for example that the testgraph in N<sub>3</sub> is modified due to changes in the informal requirements. The mapping by itself cannot accommodate such impact but has to be recreated by the tester. The mapping identifies corresponding entities in the artifacts, i. e. there is no change in the notation, excluding therefore also bridge, leading to the conclusion that we observe a *connection* mechanism. Dyad N<sub>6</sub>-N<sub>7</sub> is linked again by a *transformation* mechanism since the test sequences are generated and reflect the information in the testgraph mapping (N<sub>6</sub>).

to select, prioritize and validate requirements. Test scenarios may be developed at the same time, but require a different set of knowledge in order to realize their purpose. When information is aligned between two nodes, the context of the nodes differs and hence also the purpose of the information.

In Section 3.2.2 we have motivated how a link between two nodes can be characterized by a medium. The concept of a medium is however not able to explain how the information between two nodes is synchronized, i. e. how the change in purpose is supported by the link. Therefore we use the concept of mechanism to further characterize the link in information dyads.

To understand the mechanism concept we need to refine our earlier definition of information as *a coherent collection of related data that is created during software development, often with a specific purpose in mind*. Although this definition helps to identify nodes, as discussed in Section 3.2.1, it does not provide the granularity to differentiate between alignment mechanisms. We adopt therefore a definition in which information has the components of well-formed data *and* meaning [Floridi, 2010]:

1. data is well-formed if it has an underlying structure, syntax and notation
2. data is meaningful in a certain context, i. e. the meaning of data may change with its purpose

Using these components of information, we can now differentiate between alignment mechanisms and characterize them according to the means through which the synchronization and agreement of information, shared between nodes, is achieved.

**Transformation:** Information, packaged for one node in the alignment dyad, is re-packaged in order to satisfy the needs of the other node. A transformation mechanism that restructures and/or augments the information is applied, changing the notation and supporting the change in meaning of the data. Example: A method allows the transformation of a use case into a test model, changing the notation of the information. The support in adapting the meaning is given, for example, if relationships to other use cases are pertained in the transformation and reflected in the model<sup>1</sup>. We say that the alignment between nodes is internalized in the mechanism.

**Bridge:** Information pertaining to each node is connected and augmented in order to achieve fitness of purpose in both nodes, changing the notation. The difference to transformation is that a bridge does not provide support to adapt the meaning of data within the context change. Example: A method allows the transformation of use cases into a test model, changing the notation of information, however without establishing relationships within the test model that reflect the relationships within the use cases. Adding a new use case to the test model is supported syntactically, but the positioning in the test model requires some knowledge which is not provided by the method. We say that the alignment between nodes is semi-internalized in the mechanism.

**Connection:** Information pertaining in each node is connected, establishing a logical link between the two nodes. The mechanism does however not change the notation, nor does it provide support in adapting the meaning of the data when changing the context. The difference to the above is that the connection does not add anything to the information's fitness of purpose, except establishing a correspondence of the data component of information. Example: A method allows to link use cases to the corresponding parts of a test model, without however providing syntactical support. The meaning of the information within the test model is given only by the connections back to the use cases. We say that the alignment between nodes is not internalized in the method.

**Implicit connection:** Information is connected by volatile and implicit links that are not formalized. Such volatile links can be established by communication between people or they exist within a shared, commonly agreed upon, model. As such, it is not evident which of the components of information are effectively manipulated in a context change.

Note that the alignment mechanisms stated above are characterized by their support in preserving the relationships between information across contexts and not by their degree of automation. None of the alignment mechanisms implies that the mechanism is or can be automated.

---

<sup>1</sup> Support in adapting the meaning = preservation of relationships between information across contexts

### Example: Step 3 – In which environment is the method situated?

The table below summarizes the context of the classified method by [Miller and Strooper \[2010\]](#).

| Aspect          | Description   |
|-----------------|---|
| Method setting  | implementation of a subset of the GSM specification (<1 KLOC), focus on functional requirements, model-based testing, bespoke requirements, natural language requirements and GSM standard specifications |
| Focus           | 2) Unintentional but noted effect on alignment <sup>1</sup>   |
| Motivation      | None given due to unintentional focus   |
| Assumptions     | The specification (language) is executable  |
| Quality targets | Not stated  |
| Validation      | Testgraphs are reviewed for correctness and completeness, testgraph coverage of specification is measured   |
| Outcome         | Cost-effectiveness comparable to other model-based techniques, better than manual testing   |

<sup>1</sup> Focus is unintentional since their goal was to improve efficiency by reusing the testgraph in specification *and* implementation testing. The testgraph concept is interesting from the alignment perspective, since it is independently derived and hence an alternative representation to formal specifications of the informal requirements.

### 3.3 Method Context

In the previous step we focused on characterizing information dyads in a rather detailed manner by describing their components. In this step we broaden our view and study the context in which the described method is embedded. [Petersen and Wohlin \[2009\]](#) argue that context influences the conclusions drawn when integrating evidence from industrial studies. In a classification effort it is hence important to capture the context of the classified objects. In the following paragraphs we illustrate the context aspects that should be captured.

**Method setting:** Describe type of development process, scale/size (of the project in which the method was applied), focus of requirements types (functional, quality, both), type of testing (unit, integration, system, acceptance, formal verification, scenario-based, etc.), and type of requirements engineering (market-driven or bespoke, use of natural language primarily or other notation).

**Focus:** Describe the degree to which alignment of RE and ST is the primary focus of the method. Is an alignment issue between RE and ST thematized and addressed (choose 3, 4, or 5)? Are the studied methods/activities embedded in a software engineering problem that includes, but does not exclusively discuss RE and ST alignment (choose 1, 2, or 3)?

1. Unintentional and undiscussed/unnoted effect on alignment
2. Unintentional but noted effect on alignment
3. Part of purpose was to improve/affect alignment
4. Main purpose was to improve/affect alignment
5. Intended, main as well as sole purpose

**Motivating problem:** Describe the driver/intention/motivation to propose/implement an alignment method.

**Assumptions:** Describe any constraints or assumptions, e. g. on existing artifacts or application domains, that the application of the alignment method makes.

**Quality targets:** What is aimed to be improved by a better RE and ST alignment? Examples are reducing time-to-market, test effort, cost, number of faults, etc.

**Validation:** Is there any formal or informal mechanism that supports the consistency of the shared information? In particular, does the alignment method provide any support in assessing/verifying the consistency or correctness of the shared information?

**Outcome/Benefits:** What are the experienced effects of the alignment method? Note that this should only contain actual (not expected ones) effects that were established by an evaluation.

### 3.4 Dyad Structure Properties

The central unit of analysis of the REST taxonomy is the information dyad (Figure 3.1a). As we have illustrated in the examples in Section 3.2, a REST alignment method may consist of several dyads, thus forming a structure that is governed by the components of a dyad (Figure 3.1b). We have defined a set of six properties based upon the characteristics of nodes and links, explained in this section and illustrated in the example for Step 4. The most basic property is the number of nodes in a dyad structure. Other properties are derived from the purpose of a node, i. e. in which development phase it predominately exists, or the alignment mechanism of the link between two nodes. The definition of these properties is guided by their usefulness in interpreting and analyzing a dyad structure. In Section 3.5 we propose a classification of alignment methods based upon dyad structure properties.

#### 3.4.1 Number of Nodes ( $P_1$ )

Links between nodes need to be established and maintained over time. Hence, the total number of nodes allows one to reason on the (visible, explained) complexity, and on the effort to establish and maintain REST alignment. A large number of nodes may indicate a high cost in institution-

alizing alignment. Furthermore, even though a larger number of nodes can break down the alignment process into manageable sub-tasks, the overall complexity of the method increases with the number of nodes, as linking mechanisms between the nodes need to be defined and instantiated.

#### 3.4.2 *Branches (P<sub>2</sub>)*

Looking at an individual dyad, one node acts as a source, the other as a sink of information. A branch exists, if the dyad structure is configured such that a node acts as a source or sink for more than one node. We provide in the example for Step 4 a procedure to identify branches in a dyad structure.

Branches may reduce the complexity of analyzing information (concern separation) in sink nodes. However, at the same time branching requires a step in which the individually analyzed information is merged, introducing more nodes, potentially more effort and an increase of the overall methods' complexity.

#### 3.4.3 *Intermediate Nodes (P<sub>3</sub>)*

Nodes characterized by information that belongs to the design/analysis or implementation phase of software development are intermediate nodes. Their existence indicates that the method user is required to have knowledge outside the RE and ST domain. Intermediate nodes may strengthen overall REST alignment by integrating analysis/design and implementation, increasing the traceability. However, intermediate nodes can also imply that the method may be more invasive to the overall development process.

#### 3.4.4 *RE and ST Node Proportion (P<sub>4</sub>)*

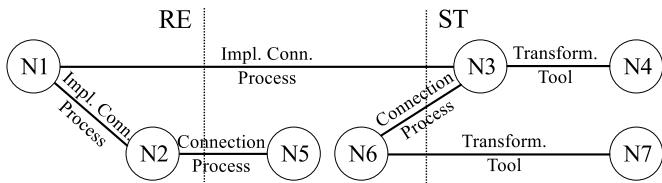
Assuming that a node is associated with a certain cost (e.g. establishing/-maintaining the information therein and links in between), it is of interest to know the node distribution among the RE and ST phases. Such an evaluation may show which phase is impacted the most by a method. Having more within-phase nodes (and links) in RE may be beneficial as the level of abstraction can be adjusted to a level that facilitates the alignment with ST. On the other hand, nodes (and links) in RE need to be efficient as requirements may change and may be refined continuously, promoting less nodes in the RE phase.

#### 3.4.5 *Within-RE (P<sub>5a</sub>)/Between-Phase (P<sub>5b</sub>)/Within-ST links (P<sub>5c</sub>)*

Based upon the information characterizing a node, we can approximate roughly its primary development phase and whether it is located early or

Example: Step 4 – What is the structure of the identified dyads?

The figure below illustrates the dyads that were identified in the method presented by Miller and Strooper [2010], using the data gathered in the examples for Step 2.1, 2.2, and 2.3.



In this example we show how the dyad structure properties are derived from this data.

*P<sub>1</sub>*: This property is calculated by counting the number of nodes identified in the method, which is in this case 7.

*P<sub>2</sub>*: The dyads are N<sub>1</sub>-N<sub>2</sub>, N<sub>1</sub>-N<sub>3</sub>, N<sub>3</sub>-N<sub>4</sub>, N<sub>2</sub>-N<sub>5</sub>, N<sub>3</sub>-N<sub>6</sub>, N<sub>6</sub>-N<sub>7</sub>, whereby the first node represents the source, and the second node the sink of information. From this sequence, we can identify the number of branches by counting the dyad instances where a source or sink node occurs more than once. In this example, this is the case for 2 source nodes (N<sub>1</sub>, N<sub>3</sub>), leading to the conclusion that we observe 2 branches in this method.

*P<sub>3</sub>*: Nodes N<sub>5</sub> and N<sub>6</sub> are intermediate nodes, hence the value for this property is 2.

*P<sub>4</sub>*: The RE and ST node proportion is 2 (N<sub>1</sub>, N<sub>2</sub>) : 3 (N<sub>3</sub>, N<sub>4</sub>, N<sub>7</sub>).

*P<sub>5a/b/c</sub>*: This property is extracted by listing the link mechanisms in the respective phases. The only Within-RE link is an implicit connection, and the Between-Phase links are implicit connection, two connections, and a transformation. The only Within-ST link is a transformation.

*P<sub>6</sub>*: For this property, we look at the nodes that act exclusively as source and sink of information in RE and ST respectively. In RE, N<sub>1</sub> is the only node that acts exclusively as source (N<sub>2</sub> is both sink and source). The information in N<sub>1</sub> is informal requirements, which can be regarded as information pertaining to early RE. In ST we have N<sub>4</sub> and N<sub>7</sub> that act both exclusively as sinks. Both contain test sequences (for specification and implementation tests respectively), which can be seen as information pertaining to late ST. Hence the scope for this method is Early RE – Late ST.

late in that phase. This allows us to reason upon the linking mechanisms within the RE and ST phases, and between those phases. The reason for such a distinction emerges from the assumption that each phase has different properties that need to be taken into account by the applied linking mechanism(s). For example, Within-RE links may need to accommodate frequent requirement changes, informally specified requirements and different requirement abstraction levels [Gorschek and Wohlin, 2006]. Within-ST links typically link test cases on different abstraction levels, whereas Between-Phase links require a more complex mapping since the context in the phases differs.

#### 3.4.6 *Scope (P6)*

By approximating the location of nodes in development phases, we can distinguish between early and late nodes. The distinction between early and late requirements is often made to differentiate between an “understanding” and “describing” phase in RE (e.g. in the Tropos development methodology [Mylopoulos and Castro, 2000]). Similarly, one can also distinguish between early and late phases in ST. For example in RE, early artifacts can be natural language requirements and use case descriptions, whereas requirements models can be put closer to the Analysis/Design phase. Similarly, test scenarios, abstract test cases and test plans can put on the left, executable test cases on the right spectrum in ST. This allows us to reason upon the scope of an alignment method with respect to the RE and ST phases, and its implications. For example, a method may not provide a link between natural language requirements and more formalized models. In a scenario where such a link would be beneficial, the method may need to be extended or combined with other approaches.

### 3.5 *Method Classification*

Up until now we have illustrated the REST taxonomy from the viewpoint of a single alignment method, that is, describing the process of identifying information dyads, extracting the context in which the method is applied/used, and characterizing the method through dyad structure properties. In this section we expand this view by proposing a classification schema for alignment methods, based upon dyad structure properties.

#### 3.5.1 *Overview of the Classified Methods*

We have applied the taxonomy, in total, on 13 alignment methods. In the remainder of this chapter they are referenced as cases A-M: A [Güldali et al., 2011], B [Flammini et al., 2009], C [de Santiago Júnior and Vijaykumar, 2012], D [El-Attar and Miller, 2010], E [Miller and Strooper, 2010],

F [Nebut et al., 2006], G [Conrad et al., 2005], H [Abbors et al., 2009], I [Damian et al., 2005], J [Arnold et al., 2010], K [Zou and Pavlovski, 2008], L [Siegl et al., 2010], and M [Metsa et al., 2007].

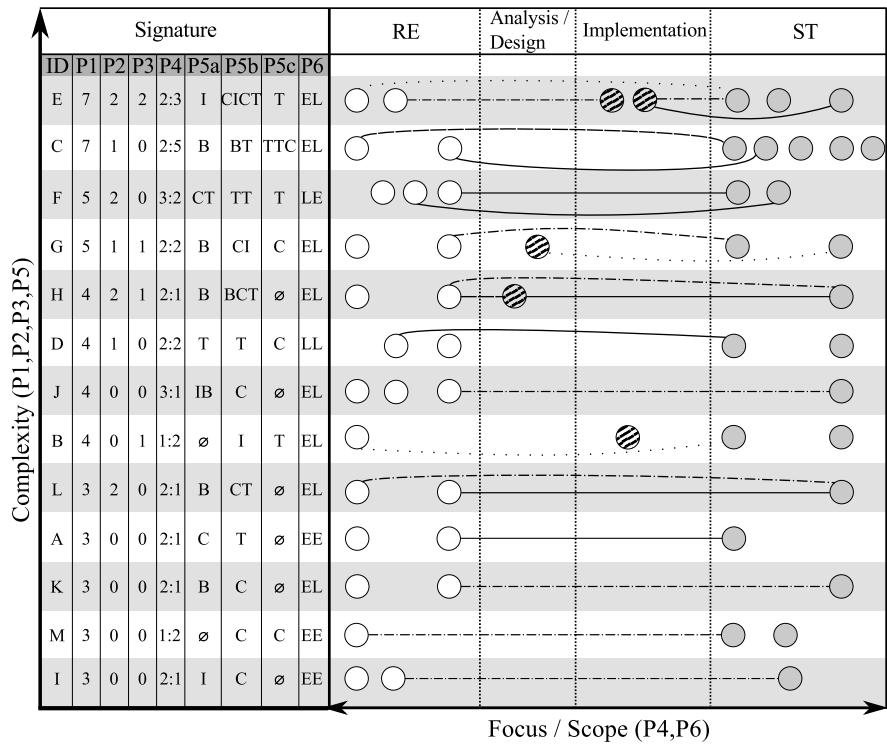
Cases F-M stem from the set of papers that were used for taxonomy construction, whereas cases A-E stem from a search in literature, as explained in Section 4.1. Since the identification and characterization of information dyads is a crucial step in the application of the taxonomy, we provide additional examples of this process on cases A-D in the appendix on page 132 (case E has served as a running example throughout this section).

### 3.5.2 Classification Schema

The schema we adopt aims at providing a meaningful and useful classification of alignment methods. Looking at the definitions of the dyad structure properties in Section 3.4, we can observe that properties P1, P2, P3 and P5 characterize the complexity, and P4 and P6 describe the focus and scope of the method. We chose therefore a simple two-dimensional schema that encodes the overall complexity of the classified method on the vertical and the focus/scope on the horizontal axis. Since we use multiple properties to represent complexity, we define the following order for sorting a method on the complexity dimension:

1. P1 (Number of nodes): this is the main sorting criterion as each node, through its associated information, contributes to the need of maintaining consistency (otherwise, the very purpose of the method would be violated).
2. P5b (Between-Phase links): each link that crosses a phase boundary (e.g. from RE to design, RE to ST), contributes to the overall complexity as information from different contexts is linked. We use the number of Between-Phase links as the second sorting criterion.
3. P5ac (Within-RE and Within-ST links): linking information on different abstraction levels is less involved than linking information from different contexts; as these links lie within one phase, we use the number of Between-Phase links as third sorting criterion.
4. P2 (Branches): even though related to the number of links, branches are less indicative for the overall method complexity as they act locally (i.e. within a dyad) as an agent to reduce complexity. They are therefore the fourth sorting criterion.
5. P3 (Intermediate nodes): everything else being equal, intermediate nodes are the fifth sorting criterion.

Note that the ordering above considers only complexity defined by the properties we have identified. For example, we do not classify the information in a node itself. Hence, the complexity of an alignment method, as defined by the classification schema, is an approximation that can be



#### Legend

##### Signature encoding:

- ID: Case identifier
- P1: Number of nodes
- P2: Branches
- P3: Intermediate nodes
- P4: RE:ST proportion
- P5a: Type of within-RE links
- P5b: Type of between-phase links
- P5c: Type of within-ST links
- P6: Scope (EE → Early RE / Early ST  
EL → Early RE / Late ST  
LL → Late RE / Late ST  
LE → Late RE / Early ST)

RE Nodes



Between Nodes



ST Nodes



Transformation (T)



Bridge (B)



Connection (C)



Implicit connection (I)



Figure 3.2: Cases classified in the REST taxonomy

improved by a more fine-grained characterization of a node's information component. As a consequence, the presented classification does *not* provide any statement on the performance of the classified alignment methods. Nevertheless, the qualitative classification of the method context (see Section 3.3), in particular the method setting, assumptions and quality target aspects, provide means to interpret and judge the quantitative classification.

The second dimension of the schema (horizontal axis), characterizes the methods according to their focus (P4) and scope (P6).

### 3.5.3 Classification Results

Figure 3.2 shows the 13 classified cases. In the left-most column, we encode the dyad structure details in a signature, whereas in the right part of the figure, the structure is represented graphically (only Between-Phase links are drawn). Note that cases A, K, M and I have the same complexity according to the sorting criteria defined in Section 3.5.2. We analyze the results of the classification in Section 5.

## 4 TAXONOMY CONSTRUCTION AND VALIDATION METHOD

In this section we describe how we constructed and validated the taxonomy, and discuss threats to validity of this approach.

### 4.1 Iterative construction and validation

In Section 2.5 we motivated why the taxonomy was constructed in a bottom-up fashion. We started by sampling alignment methods published in literature. The initial sample consisted of 16 publications that were analyzed in a systematic mapping study on aligning requirements specification and testing [Barmi et al., 2011]. Since the mapping study had limitations, as further discussed in Section 4.2.1, we added 10 more publications that we regarded relevant by reading title and abstract. Hence, the taxonomy construction pool consisted of 26 publications, from which 15 were used in the taxonomy construction process (iteration 1-4). Although we used all 15 publications in the construction process, we classified 8 of them (cases F-M) and excluded 7 for the following reasons:

- the publication covered only the RE aspect, leading to the decision that the method, described in this particular study, is out of scope: 4 publications (Hayes et al. [2006], Grunske [2008], Mugridge [2008], Niu et al. [2009])
- the publication only sketched a solution proposal or reported lessons learned, and was therefore not descriptive enough to warrant a clas-

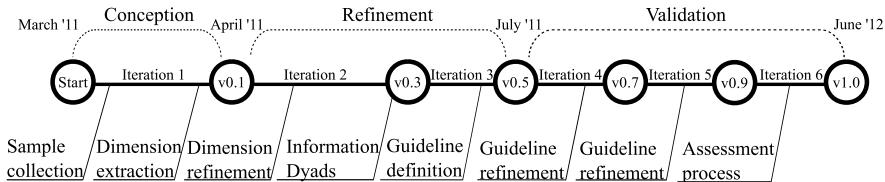


Figure 3.3: REST Taxonomy construction and validation process

sification: 2 publications (Winbladh et al. [2006], Kukkanen et al. [2009])

- the publication is a predecessor to a publication that has already been classified: 1 publication (Nebut et al. [2004])

In iteration 5 we classified 5 more publications (cases A-E) that were not included in our initial pool, resulting in a total of 13 classified methods.

Three researchers were involved at different stages in the construction and validation of the taxonomy. The milestones of this iterative process are illustrated in Figure 3.3. We discuss these iterations in the following subsections.

#### 4.1.1 Iteration 1

In the first iteration we chose five publications to bootstrap a set of dimensions for the classification of alignment methods. The first author applied open coding [Robson, 2002] on each method description and consolidated the emerged codes into dimensions characterizing the alignment approaches (v0.1 of the taxonomy, see Figure 3.3).

The strategy was to identify commonalities or distinguishing aspects in the described methods. For example, one common aspect was that information from the requirements engineering phase is reused in downstream development and eventually in system or acceptance testing, leading to a dimension describing information source and sink. Another early dimension, describing the packaging of information (e.g. in natural language, diagrams, etc.), characterized whether information is used “as is” or if it is adapted or enriched for downstream use.

#### 4.1.2 Iteration 2

In the second iteration, the second author was invited to verify whether the identified dimensions characterize the methods in an useful manner. We re-used the publications from iteration 1. Although the definitions of the dimensions were refined in this iteration, we realized that a characterization of the heterogeneous set of methods would not be possible with

static dimensions describing the method as a whole. For example, methods could be characterized by several information sources and sinks. Hence we introduced the concept of information dyads which allowed us a more fine-grained and flexible characterization, leading to v0.3 of the taxonomy.

#### 4.1.3 *Iteration 3*

The first and second author chose five new publications for the third iteration. The dimensions, now consolidated in the information dyad construct and the context aspects, were further refined and a guideline was developed (v0.5 in Figure 3.3). We chose two additional publications and exemplified the application of the taxonomy in the guidelines.

#### 4.1.4 *Iteration 4*

We invited the third author to validate the updated taxonomy and the operational guidelines developed in the previous iteration. We chose three new methods from the sample set and all three authors independently applied the taxonomy. We analyzed the results in a post-mortem.

On method 1, we achieved in general a good agreement, having however some variance on the identified medium (link characteristic) and the level of focus on alignment (context aspect). Looking at the guidelines, we identified the definitions of the different media and alignment focus levels as a cause for the disagreement and clarified them. On method 2 and 3 we observed a larger variance among the three analysts. The major reason was a disagreement on whether the method is in the scope of the taxonomy, i. e. if it can be classified as an alignment method. Hence we added Step 1, identifying the relevance of the studied method, in the taxonomy application process (see Table 3.1). The intention of the scope criterion is to clarify that we are interested in classifying methods that consider both requirements engineering and testing aspects. Methods that bridge other gaps, e. g. between design and test, are by this definition excluded.

#### 4.1.5 *Iteration 5*

The aim of this iteration was to apply the taxonomy on a set of methods that were not included in the initial set of publications. To this end, we chose four premium venues for publications in Requirements Engineering (Requirements Engineering Journal), Verification & Validation (Software Testing, Verification and Reliability) and software engineering in general (IEEE Transactions on Software Engineering, Software Quality Journal) as the population for drawing our sample. We chose 2007 as the starting point for our search since we did not aim to perform a systematic literature review [Kitchenham and Charters, 2007] and hence do not claim complete time coverage. Furthermore, 2007 seemed to be a good starting point since

Cheng and Atlee [2007] and Bertolino [2007] called for a closer collaboration between requirements engineering and software testing research at FoSE that year.

The first author manually searched, by reading title and abstract, 635 publications from the period 2007–2011, applying the criteria defined in Section 3.1. After applying the scope criterion on the abstracts, 148 publications remained for full text screening. In this step, the scope criterion was applied a second time, excluding methods that were only partially bridging the gap between RE and ST, e. g. verification of UML models [Siveroni et al., 2010], derivation of specifications from requirements [Seater et al., 2007], or derivation of test cases from design artifacts [Pickin et al., 2007], leading to 24 publications<sup>2</sup>. On these, we applied the comprehensiveness criterion, including only those methods for which we could answer the questions posed in steps 2 and 3 of the taxonomy application process (see Table 3.1). We concluded the search with 5 publications describing alignment methods and applied the taxonomy, leading to two further refinements to the guidelines:

- Introduction of use-relationships between nodes. For example in Case B (see Table 3.6), node N<sub>3</sub> contains information that is necessary for the method, is however not related with any other node through a link mechanism. The use-relationship legitimates N<sub>3</sub> in the dyad structure, increasing the richness of the method characterization.
- Introduction of a further aspect in context identification (Step 3) of the taxonomy process (Section 3.3). Recording assumptions or constraints helps to understand under which circumstances a method may be applicable.

The results of the taxonomy application on four cases (one method has served as the running example in Section 3) are illustrated in the appendix on page 132.

#### 4.1.6 Iteration 6

The aim of this iteration was to evaluate whether the REST taxonomy provides support in identifying misalignment in a development organization. The first author developed an assessment guideline and procedure, REST-bench, that is powered by the concepts underlying the REST taxonomy. The approach and assessment results are described in Section 6.

---

<sup>2</sup> In the title and abstract screening we were rather inclusive, resulting in many irrelevant studies in the set for full text reading.

## 4.2 Validity Threats

The bottom-up construction of the taxonomy is subject to several validity threats [Wohlin et al., 2000].

### 4.2.1 Internal Validity

The systematic mapping study by Barmi et al. [2011], from which we sampled publications and bootstrapped the dimensions of the taxonomy, was initially designed to identify alignment methods focusing on *non-functional* requirements and software test. Although the scope has been extended to include also functional requirements, the mapping study may have missed relevant studies<sup>3</sup>. We added therefore 10 studies that we considered relevant. Still there is a moderate threat that our sample of methods was biased.

### 4.2.2 Construct Validity

The identification of characteristics defining an alignment method, as described in Section 4.1, is subject to mono-method bias [Wohlin et al., 2000]. The first author performed the initial analysis and may have subjectively biased the taxonomy construction. To counteract this threat, we designed the taxonomy construction as an iterative process, involving multiple researchers with expertise in both requirements engineering and software verification & validation.

### 4.2.3 External Validity

During the validation we performed a manual search on four premium journals, identifying further methods and applying the taxonomy. The selection was based on reading the title and abstract of the study, searching for indications that both requirements engineering and software testing aspects were discussed. This means that “partial” solutions that bridge for example the gap between user requirements and requirements specifications (e.g. Liu [2009]), requirements to design (e.g. Valderas and Pelechano [2009]), or design to test (e.g. Samuel et al. [2007]) were not considered to validate the taxonomy.

The goal was to validate whether the taxonomy can be applied on alignment methods that were not part of the construction sample and not to identify and classify all existing methods. A thorough overview of alignment methods could be performed by conducting a systematic literature review [Kitchenham and Charters, 2007]. The review could be designed to

---

<sup>3</sup> Two studies ([Flammini et al., 2009, El-Attar and Miller, 2010]) that were identified in the manual search during the validation were not identified by the search (November 2010) in the mapping study.

include the type of the above mentioned solutions, and, by using the taxonomy presented in this chapter as an analysis aid, provide practitioners support in selecting and combining methods, as well as provide researchers an overview for further empirical or conceptual research.

## 5 METHOD EVALUATION USING THE REST TAXONOMY

In this section we elaborate on the application of the taxonomy, exemplifying analysis on two levels. First, we show the potential of the taxonomy as a mean to describe the state-of-the-art of REST alignment methods in Section 5.1. Then, in Section 5.2 we illustrate the application of the dyad structure property analysis introduced in Section 3.4.

### 5.1 Summary Analysis

In Figure 3.2 on page 110 we have classified the alignment methods presented in the 13 studied cases which allows us to perform basic quantitative analysis. We observe that the mode for number of dyads is 2, the median is 3. This indicates that methods with more than 4 dyads are uncommon. A similar observation can be made on the number of nodes, with a mode of 3 and a median of 4. Methods with more than 4 nodes are not common.

The right part of Figure 3.2 shows the distribution of nodes in the respective software development phases. The links between nodes highlight dyads which span over distinct development phases. Overall, we can observe a slight majority of nodes in the earlier phases (RE:26, ST:24). This tendency is more pronounced (RE:17, ST:12) if we exclude the cases C, E, F and G, which have an untypical (w.r.t. the mode) high number of nodes.

Looking at the alignment mechanisms, *connection* and *transformation* are the most common alignment mechanisms with a frequency of 15, followed by *bridge* (9) and *implicit connection* (6). The proportion of within and between phase links is 1:1, i. e. there are 22 links between and equal as many links within development phases. Figure 3.2 illustrates also the types of mechanisms linking nodes in distinct development phases (within-phase links are not shown). Overall, we observe that for the *connection* mechanism the between-phase links dominate (9 out of 15), whereas for the *bridge* mechanism within-phase links dominate (7 out of 9). For the *transformation* and *implicit connection* mechanism, within- and between-phase links are equally distributed (7 within, 8 between and 3 within, 3 between). The occurrences of alignment medium are as follows: *Process* (22), *Tool* (17), *Structured artifact* (4) and *Organization of work environment* (1).

The *connection* mechanism, which we defined as establishing a logical link between information in two nodes (see Section 3.2.3), can be viewed

as a mean to establish traceability. Given that this alignment mechanism, together with *transformation*, was observed most frequently, we can assert that establishing traceability is, in general, a main concern of the studied alignment methods. As shown in the analysis, the between-phase links with a *connection* type mechanism dominate, mapping for example technical requirements to test scenarios (Case I [Damian et al., 2005]), requirements classification trees to logical test scenarios (Case G [Conrad et al., 2005]), or test reports to requirements models (Case H [Abbors et al., 2009]). This observation concurs with Gotel and Finkelsteins' [1994] definition of requirements traceability referring to "*the ability to describe and follow the life of a requirement, in both forwards and backwards direction*". Note however that traceability (respectively nodes) to the analysis/design and implementation phase is sparse due to our selection criteria for RE and ST alignment methods (we excluded methods which addressed only a subset of the development phases). One exception is Case E [Miller and Strooper, 2010] in which formal specifications and testgraphs are mapped to the implementation.

In the analysis we have identified eight between-phase links featuring a *transformation* mechanism. Looking at Figure 3.2, the corresponding nodes are almost exclusively (except Case D [El-Attar and Miller, 2010]) located in the late RE phase, preceded by one or more nodes. This pattern is expected for model transformations, e.g. as in Case F Nebut et al. [2006] (use case transitions system → test objectives) or Case L [Siegl et al., 2010] (time usage model → test cases). It also shows that transformation links from early RE phases to ST are not common.

One aspect that is currently not considered in the taxonomy is the cost of creating and maintaining the links between nodes and hence maintaining the alignment. Would the taxonomy have been available to the originators of the discussed alignment methods, they could have assigned a relative cost to each link. That would allow us to compare the cost of the methods in the distinct software development phases. Furthermore, an absolute cost measure would allow one to reason on return on investment [Unterkalmsteiner et al., 2012], provided that the benefits can be estimated too.

## 5.2 Dyad Structure Analysis

The goal of this analysis is to provide means to reason on the benefits and liabilities of REST alignment methods. In particular, the analysis allows to discuss the trade-offs of methods on a level that is relevant for practitioners that seek to adopt a method and to improve REST alignment in their context. The trade-off analysis is based upon the dyad structure properties defined in Section 3.4.

For each of the properties, a value can be extracted from the dyad structure that has been established when applying the taxonomy on the REST alignment method. Then, benefits and liabilities can be elaborated for each dyad structure property. Table 3.2 illustrates this analysis on four methods, using the results from the taxonomy application shown in Section 3.5.

The abbreviations in the first column in Table 3.2 refer to the dyad structure properties defined in Section 3.4: P1 (Number of nodes), P2 (Branches), P3 (Intermediate nodes), P4 (RE and ST nodes proportion), P5a (Within-RE links), P5b (Between-Phase links), P5c (Within-ST links), P6 (Scope). The values in the second column are based on the results of the taxonomy application illustrated in the appendix on page 132.

Table 3.2: Example of trade-off analysis using dyad structure properties

| Property | Value                       | Benefit   | Liability  |
|----------|-----------------------------|---|--|
| Case A   |                             |   |  |
| P1       | 3                           | Few artifact types involved   | Transf. in dyad N2-N3 complex and iterative                          |
| P4       | 2:1                         | Reduces ST effort   | Limited to abstract test cases                                       |
| P5a      | Connection (N1-N2)          | Efficient for new/changed requirements  | None   |
| P5b      | Transformation (N2-N3)      | Defined and repeatable process  | Relyes on specific notation for requirements                         |
| P6       | Early RE – Early ST         | Supports ST in defining test scope  | Concrete test cases are not created                                  |
| Case B   |                             |   |  |
| P1       | 4                           | No new artifact types are introduced  | Tailored for a specific reference architecture                       |
| P3       | 1                           | Supports the semi-automated generation of test cases  | Incorrect system configuration may cause faulty executable tests     |
| P4       | 1:2                         | None  | Abstraction level not easily matched                                 |
| P5b      | Implicit connection (N1-N2) | Given natural language requirements (NLR's) are appropriately formulated, mapping to abstract test cases is straightforward | Mapping is not explicit; domain knowledge required to create mapping |
| P5c      | Transformation (N2-N4)      | Instantiation of abstract test cases for a specific configuration   | Correctness of configuration itself is not verified                  |

Continued on next page

**Table 3.2 – continued from previous page**

| Property | Value   | Benefit  | Liability  |
|----------|---|--|--|
| P6       | Early RE – Late ST  | Tests cover requirements considering specific configurations   | Early link (N1-N2) does not address different abstraction levels of NLR's and test cases |
| Case C   |   |  |  |
| P1       | 7   | Broken down complexity into simple steps   | Artifacts needed solely in testing   |
| P2       | 1   | Separation of concerns (Scenario development / Statechart model)   | Information needs to be merged again for testing purpose                                 |
| P4       | 2:5   | Analysis of reqs. tailored to support testing  | Limits reuse in other development phases   |
| P5a      | Bridge (N1-N3)  | Enables transformation for between-phase link (N3-N5)  | Domain knowledge required to establish and maintain                                      |
| P5b      | Bridge (N1-N2) / Transformation (N3-N5)                     | Formalized and automated transformation  | Transformation depends on three previous links   |
| P5c      | Connection (N2-N4) / Transformation level... (N5-N6, N6-N7) | Step-wise refinement and ...except for N2-N4, which may introduce a bottleneck when scenarios or SRSs change |  |
| P6       | Early RE – Late ST  | Enables traceability, allowing to verify requirements coverage   | Although partly automated overall, nodes in early RE are linked manually                 |
| Case D   |   |  |  |
| P1       | 4   | Few newly introduced artifact types  | None   |
| P2       | 1   | Enables link between problem and solution domain   | Needs to be maintained in parallel as requirements change to avoid inconsistencies       |
| P4       | 2:2   | Similar abstraction level in both RE and ST  | None   |
| P5a      | Transformation (N1-N3)                                      | Defined and structured process   | Requires training to apply correctly   |
| P5b      | Transformation (N1-N2)                                      | Usable even without executable test cases  | Uses information from different models, potentially causing inconsistencies              |
| P5c      | Connection (N2-N4)  | Enables traceability   | None   |
| P6       | Late RE – Late ST   | Focus on artifacts that have similar abstraction level   | Does not cover early RE, e.g. natural language requirements specifications               |

The current set of dyad structure properties defines four properties that can, by their nature, be found in every REST alignment method: each

method consists of two or more nodes (number of nodes (P1)), of which one or more nodes belong either to the RE or ST development phases (RE and ST nodes proportion (P4), between-phase links (P5b) and scope (P6)). As such, these four properties underline the scope criterion of alignment methods described in Section 3.1 and hence define a minimum set of properties for a REST alignment method.

The remaining properties (branches (P2), intermediate nodes (P3), within-RE and within-ST links (P5a, P5c)) are not featured by every alignment method, as seen for example in Case A in Table 3.2. They do however provide relevant information on the alignment methods as the benefits and liabilities show in Cases B, C and D. Concluding on the dyad structure analysis, the six properties provide means to characterize and analyze individual REST alignment methods, are however not adequate to enable a comparison between methods as not all properties can be observed in every method. The assessment of benefits and liabilities in Table 3.2 should therefore be interpreted in the context of the respective methods. For example, the methods presented in Cases A and B, with a relatively low complexity according to our classification, rely on a certain requirements specification form and reference architecture (see assumptions in Table 3.3 and Table 3.5). Furthermore, the motivations and targeted goals of these methods differ (test process efficiency vs. test coverage), such that general conclusions on the adequacy of a method, based alone on the quantitative classification of dyad structure properties, are likely not to be accurate. In order to reduce the risk of a misleading taxonomy application, we recommend therefore to interpret the quantitative classification in conjunction with the qualitative classification (method context), which provides information that indeed allows adequacy judgments on a method with respect to particular company settings and goals.

### 5.3 Lessons Learned and Limitations

In Iteration 5 of the taxonomy construction (see Section 4.1) we searched in 635 publications for REST alignment methods. We expected to identify a number of publications that would allow us to illustrate the characteristics of the state-of-the-art REST alignment methods. We excluded however, applying the scope criterion (see Section 3.1) 630 publications, indicating that there is a lack of research and solution proposals on supporting the alignment between RE and ST. On the other hand, we identified several “partial” solutions (from the RE and ST alignment perspective), that address specific gaps. From the RE perspective we observed efforts to improve the traceability from requirements engineering activities and artifacts to design (e. g. [Navarro et al., 2010, Houmb et al., 2010, Pires et al., 2011]), and similarly, from the ST perspective, test generation from design artifacts (e. g. Xu et al.

[2012], Kundu et al. [2009], Pickin et al. [2007]). Together with the low number of identified RE and ST alignment methods, this indicates that the envisioned closer collaboration between RE and ST researchers [Cheng and Atlee, 2007, Bertolino, 2007] is still in its early development, that there is potential in streamlining the efforts in the respective areas, and that the proposed taxonomy can indicate gaps in research. For example, it could be investigated whether the partial solutions can be combined and which adaptions need to be made to construct new REST alignment methods.

Regarding the components of the taxonomy, we experienced that the classification of medium, characterizing the link in an information dyad, can be confounded. The medium characterizes a link between nodes, not the information in the node. This makes the analysis conceptually more difficult and may lead the analyst to (wrongly) classify the medium of information in the node instead of the link. On the other hand, applying the taxonomy according to the guidelines (Section 3) and limiting the characterization of the medium on the link, leads to classifications were the medium is often a process (i.e. the process/activity transferring the information from node A to node B). A factor that contributes to the difficulty in classifying the link medium is that the taxonomy defines a medium both as a carrier of information and also as a facilitator that enables information transfer (see Section 3.2.2). Further use or application of the taxonomy might show whether medium as a characteristic of an information link needs to be refined, either by a more precise definition or by modeling it in a different manner. In this study we have tried to strike a balance between analytical depth and taxonomy usability and thus opted for not refining the concept of a medium.

The construction and application of this particular taxonomy was subject to a circular problem. The publications and RE and ST alignment methods we studied were likely not written with the concept of an information dyad in mind. Still the concept can be used to characterize a wide variety of alignment methods. Extracting the characteristics is for the same reason challenging and for some cases not objectively possible e.g. de Caso et al. [2012], Uzuncaova et al. [2010], and Grieskamp et al. [2011], which we excluded from further analysis in Iteration 5 although seemingly relevant. Would there have been a taxonomy on RE and ST alignment methods available when these methods were conceived, they may have been reported differently. We propose that our taxonomy can be used to structure and give detail in future papers that report on alignment methods. Such an effect has been observed in the Global Software Engineering community after the publication of a classification scheme for empirical research in the area [Šmite et al., 2008].

## 6 INDUSTRIAL CASE STUDY USING REST-BENCH

To make the REST taxonomy relevant for industrial assessment of alignment we created a lightweight framework. REST-bench is powered by the REST taxonomy, reusing the information dyad and dyad structure concepts presented in Section 3, but also includes process elements (how to use REST-bench) and analysis and visualization elements. This section shows the industrial application and test of the taxonomy through the use of REST-bench in a case study. We describe REST-bench in Section 6.1, present the results in Section 6.2 and illustrate the dyad structure analysis in Section 6.3.

### 6.1 REST-bench Process Overview

The goal of the assessment is to identify improvement opportunities in the coordination between the requirements engineering and the system testing organization. In order to elicit information on the current state of affairs, REST-bench focuses on the relationships between artifacts created and used by the different roles in the software organization, particularly by RE and ST roles. The choice of centering the assessment around artifacts is motivated by their importance in carrying information, which is the basis for characterizing alignment (as we have illustrated with the information dyad in the REST taxonomy). The objectives of the assessment are to:

- elicit, from the RE and the ST perspective, the artifacts that they create and for which purpose these artifacts are created
- contra-pose those two perspectives to identify disagreement
- identify deficiencies in the creation/use of artifacts that impede alignment

The procedure to achieve these objectives is summarized in the following steps.

**SELECTION (STEP 1):** Representatives from the RE and the ST role are interviewed. One important constraint for the selection of interviewees is that they have or are currently collaborating in the same project. This allows elicitation of information on the actually created and used artifacts instead of referring to what is prescribed or recommended by the official process in an organization. The interviews with the RE and ST representative are conducted separately, following a guideline that supports the analyst to collect information regarding the context of the agreed upon project and the artifacts created and used in the project.

**MAP CREATION (STEP 2):** The analyst creates an artifact map which shows use-relationships between the artifacts in the studied project.

Furthermore, each artifact is annotated with the role(s) that created and used it during the project. This artifact map merges the perspectives of the RE and ST representatives, providing a basis for discussion and analysis during the following step.

**ANALYSIS WORKSHOP (STEP 3):** The analyst, RE and ST representatives conduct a workshop in which the artifact map is reviewed. Artifacts, relationships, users and creators are confirmed, modified or extended. The RE and ST views are merged and the analyst uses the dyad structure properties to elaborate together with the workshop participants potential improvements.

We reuse the concept of dyad structure properties, introduced in Section 3.4, in the analysis of the nodes represented in the artifact map, refining however the definition of the properties to the particular context of assessing alignment (alignment-as-state). For each property, we propose a set of questions the analyst may ask during the workshop to initiate the discussion and analysis.

#### 6.1.1 Number of Nodes ( $P_1$ )

In the assessment of an organization, the number of nodes relevant for REST alignment is a first indicator for identifying bottlenecks or overhead. Too few nodes can indicate challenges in coordinating RE and ST activities since the necessary information is not shared effectively. On the other hand, too many nodes can indicate that much effort is spent on keeping these nodes up-to-date and synchronized, not to mention roles and responsibilities. Each node can represent an individual, department or role in an organization. If this is the case, each link between nodes can imply the creation of overhead and possibilities for everything from misunderstandings [Gorschek and Davis, 2008] to mis-communications due to sub-optimization [Fricker et al., 2010].

- Is there an information need that was not fulfilled by the used artifacts?
- (If applicable) Given that artifact X doesn't have any user / is only used by A, could the information in artifact X be merged into artifact Y?

#### 6.1.2 Branches ( $P_2$ )

The contribution of branches to REST alignment needs to be evaluated. In particular, whether the information in a branch node is actually used in either a RE or ST activity, or whether the branch has a different purpose. Such a distinction is needed for a more accurate estimation of the spent effort for REST alignment. In extreme cases a branch can be seen as "dead",

adding nothing to alignment, which becomes visible when applying the taxonomy evaluation, but is not obvious in every-day activities. For example, test plans that are derived from initial requirements specifications have little value if not updated and maintained as the development proceeds and requirements change. In such a scenario a detailed test plan, e. g. specifying which requirements are covered by which test cases, is waste as it won't be used, due to its inaccuracy, during testing.

- How is the information in artifact X kept consistent with the information in artifact Y, in the case Z changes (Z has two links, a branch, to X and Y)?
- If inconsistencies between artifacts X and Y arise, how does that impact the users of those artifacts and their work?

#### 6.1.3 *Intermediate Nodes (P<sub>3</sub>)*

It is important to identify and to understand the purpose of intermediate nodes for two reasons. First, changes to nodes in the design or implementation phase may inadvertently affect REST alignment. For example, the replacement of a design artifact with more frequent meetings between analysts and programmers may reduce the documentation effort, however breaking at the same time an important link that establishes a connection between high-level requirements and system tests. This is especially relevant for organizations moving from a plan-driven to a lean/agile development process where an improvement in saving of perceived overhead in terms of documentation can be a sub-optimization from a product perspective [Gorschek and Davis, 2008]. Second, intermediate nodes may also represent an overhead that may be eliminated without affecting REST alignment. Both scenarios become visible through the application of a REST alignment analysis.

- Do the creators of artifact X (designers), deliver timely, i. e. can the information actually be accessed in ST when necessary?

#### 6.1.4 *RE and ST Node Proportion (P<sub>4</sub>)*

The proportion of nodes in the RE and ST phases can be used as an indicator for the relative effort spent on REST alignment in the respective phases. A REST alignment assessment can enable an overview and subsequent optimization by removing or changing items detrimental for alignment, as seen next.

#### 6.1.5 *Within-RE (P<sub>5a</sub>) / Between-Phase Links (P<sub>5b</sub>) / Within-ST (P<sub>5c</sub>)*

The amount and quality of Within-RE, Within-ST and Between-Phase links can drive improvements that aim to optimize the links as such and their

interplay throughout the development phases. For example, an ad-hoc connection of business with user requirements could be replaced by a more rigorous mechanism that explicates relationships between business requirements that are also reflected in user requirements. Such a change in RE is however only reasonable if user requirements are actually linked to test cases, such that ST can adapt its test strategy on the information provided in RE. If the Between-Phase link does not exist or is inefficient, an improvement of the RE links would not achieve the anticipated benefits. Thus, an SPI initiative or just introducing a tool might be beneficial for requirements management, but without a REST alignment assessment the benefit or even detrimental nature of a change can not be seen.

- How does staff turnover affect the quality of requirements and derivative artifacts?
- In case requirements change, by whom/how/when are these changes propagated to linked artifacts?
- Does inconsistency of information among artifacts affect the work in: RE, ST, the interface between both?

#### 6.1.6 Scope (P6)

Scope evaluation can lead to the identification of areas in RE or ST where the alignment may be improved or even established as the taxonomy makes deficiencies in linking information explicit. Most importantly, it allows identifying gaps that would have not been perceived as such when looking at them from either the RE or ST perspective alone. For example, system requirements may be linked to system tests, allowing to determine test progress. However, if the system requirements are not linked to business requirements, a statement on the verified portion that provides business value can not be made.

- (Identify artifacts that are created by RE and ST and ask which input is used to actually develop them)
- How is the consistency between these inputs and the developed artifacts maintained over time? What are the advantages/drawbacks of maintaining this consistency?

#### 6.2 Case Study: REST-bench Results

The case presented in this chapter is based on a research collaboration with Ericsson AB, Karlskrona. Ericsson is involved in the development of embedded applications for the telecommunication domain. The studied project, completed in autumn 2011, had a duration of appropriately one calendar year. The staff consisted of 150 engineers which were split up in seven teams. The system requirements consisted of approximately 350

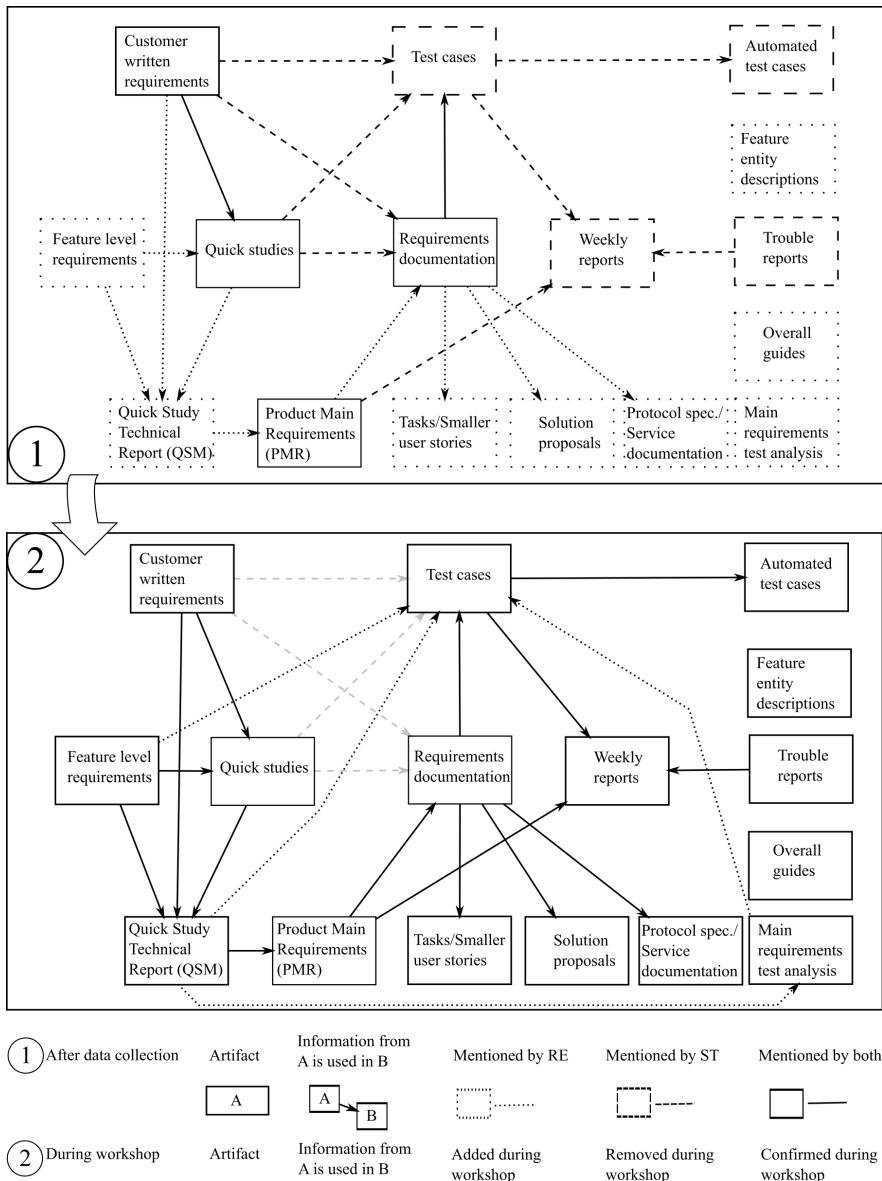


Figure 3.4: REST-bench artifact maps from the Ericsson case study

user stories. The system test cases amounted to 700, of which 550 were automated. The interviewees were selected based on their work experience and their collaboration during the project. The RE representative, a system manager, had 12 years of experience in his current role whereas the ST representative, a verification engineer, had 14 years of experience.

Figure 3.4 illustrates the artifact maps that were created after the interviews (Map 1) and during the workshop (Map 2). To improve readability, the maps in Figure 3.4 are not annotated with the creator and users of the artifacts as they were elicited in the interviews. During the workshop these annotations were however useful to discuss the purpose of certain artifacts and to identify responsibilities regarding their maintenance during the project life-cycle. The relationships between artifacts, represented by a directed arrow, indicate the transfer of information. For example in Map 1, ST uses information from “customer written requirements”, “quick studies” and “requirements documentation” to create “Test cases”. Map 1, merged from the RE and the ST perspective on the project documentation, gave rise to several observations that were addressed during the workshop:

- ST uses more artifacts to create test cases than RE is aware of
- “Feature entity descriptions” and “Protocol specifications/Service documentation”, both created by RE with ST as user, are not used by ST
- How are RE artifacts (“Customer written requirements”, “Feature level requirements”, “Quick studies” and “Requirements documentation” kept consistent when requirements change?
- “Requirements documentation”, consisting of user stories, is not traced explicitly to the corresponding test cases; a specific role, the Technical Manager for Test, mediates between test engineers and RE.

During the workshop, Map 1 was handed out to the RE and ST representatives. They reviewed each artifact and link, proposing the changes leading to Map 2. Notice that most of the changes (removed and added links) refer to the interface between RE and ST. This indicates that the co-ordination *within* RE and ST respectively is well understood, however the interplay *between* RE and ST is rather opaque when viewed from a single perspective, emphasizing the value of the process of creating artifact maps. The discussion during this process lead also to a knowledge transfer among RE and ST, clarifying several misconceptions on the use of artifacts by the different roles in the project:

**USE OF ARTIFACTS:** RE stated that service documents are rather seldom updated since they are of little use for ST. ST clarified that they are quite often used for testing. *Potential impact:* ST may rely on outdated information in service documents, leading to failed tests and/or unnecessary trouble reports.

**LIFETIME OF ARTIFACTS:** RE assumed that automated test cases (test cases for legacy functions) are linked to user stories or requirement statements. ST clarified that they are mapped to commercial features, which are part of the product specification and not of the products requirements documentation. *Potential impact:* The usefulness of the requirements documentation for ST ends to a certain extent when a manual test case is automated (the link to the requirements documentation is replaced by a mapping to a commercial feature). Assuming that requirements are reused in follow-up projects, ST requires additional effort to decide whether new test cases need to be created or automated test cases can be reused.

**INFORMATION DISPERSION:** The Technical Manager for Test (TMT) serves as a link between RE and ST and RE assumed that the TMT initiates testing, forwarding the necessary information from RE. However, according to ST, testers actually pull the necessary information from the TMT. *Potential impact:* The resources of the TMT are not efficiently used since he interacts individually with testers. Information may be available to some testers while others may not have contacted the TMT, leading to inconsistencies among the testers' knowledge about the system.

### 6.3 Identifying improvements using REST-bench

After the artifact map was reviewed by the workshop participants, leading to Map 2 in Figure 3.4, the analyst used the dyad structure properties (see Section 6.1) to guide the elaboration of improvement opportunities on the basis of the artifact map.

**P1 – NUMBER OF NODES** Looking at Map 2 in Figure 3.4, it should be observed that ST uses “Feature level requirements”, “Quick Study Technical Report (QSM)”, “Main requirements test analysis” and “Requirements documentation” as input to create test cases. The latter one is the main source, whereas the others are used complementary. The question arises whether inconsistencies in these documents, caused for instance by requirements changes during the design or implementation, affect ST. According to RE, changes in user stories (part of the requirements documentation) are propagated to the QSM, and the affected development teams and ST are invited in presentations where these changes are discussed. In the context of this particular project, according to ST, there were inconsistencies due to a too early test analysis. RE states that there was a need to redesign parts of the solution, leading to consequences which were not dealt with.

**P3 – INTERMEDIATE NODES** “Feature entity descriptions” describe the system functionality on a compound level. RE states that early creation of this document would help ST since it shows a better use-case of the system than the documentation written by the individual development teams (which may be inconsistent with respect to each other). Feature entity descriptions are written late since they describe how the system is actually implemented and to be used which is completely known only quite late in the project. To make the feature entity descriptions useful to ST, they would need to be maintained and updated during the project as the implementation stabilizes. RE states that writing the feature entity descriptions late is a local sub-optimization since other users would benefit of being able to use them. A factor that may contribute to the difficulty of maintaining feature entity descriptions is that the responsibility to write them is given to technical writers (external consultants).

**P5B – BETWEEN-PHASE LINKS** An important interface between RE and ST is the link between “Requirements documentation” and “Test cases”. There is however no mapping between requirements and test cases at the Integration Test level. According to ST, this is due do the difficulty to keep this mapping up-to-date (spreadsheets are inefficient) or to import the required information into the test management tool. According to RE this lack of mapping may lead to lower test coverage of the requirements. Looking at Map 2 in Figure 3.4, the main requirements test analysis is based on the quick study technical report. The Technical Manager for Test (TMT) serves as a link between RE and ST, performing the main requirements test analysis, and defining the scope for the testing effort. In this case, the between-phase link is implicitly established by a role, leading to the consequences discussed in Section 6.2, i. e. an inconsistency in the tester’s knowledge on the system and test scope.

#### 6.4 *Lessons Learned using REST-bench in Industry and Limitations*

Overall, the assessment can be considered as lightweight in terms of effort for both the analyst and the organization. The interviewees invested 1.5 hours each for the interview and 2 hours for the joint workshop. In this first-time application of REST-bench, the analysis of the collected interview data, including the creation of artifact maps and a report summarizing the findings, required 5 days of full-time work. We expect however that, with increasing experience, the effort for the analysis can be reduced to 2 working days.

The format of the assessment, with separate interviews and a joint workshop, allowed us to pinpoint disagreements in the perspectives of RE and ST representatives on project documentation. The test engineer stated

that “we have a need of improving the coordination between RE and ST. ST is in the end of the development phase and requirements have been changed/removed and are sometimes ambiguous. There is always a need to understand what/how/why other parts of the organization work.” The requirements engineer agreed that REST-bench could complement a project post-mortem process, stated however also that “it should then include more roles and perhaps also dig into not only which documents are used but also what [parts of information] in certain documents are actually used.”

We observed a mutual learning effect among engineers that worked for a year on the same project and for over a decade in the same organization. As such, the artifacts map and its review during the workshop is a valuable tool to create a shared understanding of the coordination mechanisms in a large development project.

The dyad structure properties served as heuristics to identify potential sources of misalignment between RE and ST. They allow for a focused analysis of issues that emerge when both RE and ST perspectives are considered. The dyad structure properties are thereby an useful abstraction of the detail in the REST taxonomy, enabling a more effective interaction and communication with industry.

In the assessment we used artifacts (their creation and use) as a proxy to elicit and understand the alignment between RE and ST. Artifacts are tangible and relatively easy to describe in terms of their purpose and content. Hence, they were a natural choice to structure the elicitation and analysis. The assessment can be however extended to achieve a more fine-grained picture of the state of REST alignment. For example, informal communication channels and information sources, such as e-mail, internal wikis, instant messaging and telephone calls, or meetings could be included in the analysis. Such a detailed assessment would however require focusing the analysis on a limited set of activities where RE and ST interact.

## 7 CONCLUSIONS AND FUTURE WORK

Taxonomies are means to structure our knowledge and to discover new relationships between concepts, advancing the understanding of observed phenomena. The taxonomy presented in this chapter aims at characterizing methods for requirements engineering (RE) and software test (ST) alignment. Although both RE and ST are mature research areas, their interplay has been less explored. Investigating these relationships, structuring and communicating them are the major contributions as summarized below:

- We have investigated the RE and ST alignment phenomenon by applying a bottom-up and iterative approach to construct a taxonomy.

The principles of this approach might also be useful to construct taxonomies in other research areas.

- The structuring of concepts that belong to different areas is a challenging task. The information dyad is an abstraction that supports the reasoning on RE and ST alignment methods. Although we expect that the taxonomy can be refined and extended, we claim that the concept of an information dyad is a valid construct to characterize RE and ST alignment in particular, but we also foresee it as being valid to characterize alignment in other domains of SE as well. Essentially, information can never be adapted or aligned to each other without some type of link between them. The dyad makes this into a first class concept and thus allows to clarify and compare links as well as the information being linked much more explicitly and formally.
- Clear definitions support communication. We have defined RE and ST alignment both as a state and as an activity, since the meaning may differ depending on the context. In the context of alignment-as-activity, we developed the REST taxonomy, identifying and describing dyad structure properties that allow one to reason upon the phenomenon of alignment. In the context of alignment-as-state, we developed REST-bench, an alignment assessment framework powered by the REST taxonomy.

The application of the taxonomy on 13 REST alignment methods allowed us to reason on the overall topology of the methods. We have observed a median of three information dyads and a tendency to have more nodes in earlier development phases, i. e. in RE, than in the ST phase. Assuming that the complexity of a method, and therefore the effort of applying it, increases with the number of nodes, this indicates that most of the classified alignment methods require a relatively higher effort in the start-up phase (RE) than in follow-up phases (Analysis, Implementation, ST). Interestingly, the two most complex alignment methods (Case E and C), according to their dyad structure properties, are geared towards a focus in ST. This is rather surprising as one would intuitively drive the alignment effort from requirements engineering activities (this intuition is confirmed by the majority of the other methods we classified). This could indicate that these two methods work better in a context where the RE activities relevant for alignment are already well understood and established.

On the opposite end of the spectrum, we observed four alignment methods (Case A, K, M and I) that were classified with a similar, low complexity, differing however in their focus and scope. However, only Case K has a scope that reaches from early requirements engineering to late testing. This indicates that the complexity of alignment methods correlates with

their scope (although we observed also an exception with Case F), which is not surprising.

Based on the manual search in 635 publications we identified only five REST alignment methods. One explanation for this low number may be the strict inclusion criteria, i.e. that the publication fulfills the relevance criteria (scope, comprehensiveness, rigor) stated in Section 3.1. The scope criterion turned out to be the most selective as it excluded a wide range of publications that did not target *both* the RE and ST areas (see Section 4.1.5). Even though the analysis of the studied methods revealed that the *connection* mechanism, which enables traceability, was found most frequently, the overall low number of identified REST alignment methods signals for more research and solution proposals that aim to bridge the gap between RE and ST.

We developed an assessment framework with the REST taxonomy as engine, called REST-bench, and applied it in an industrial assessment at Ericsson AB. The concepts of the REST taxonomy, integrated in REST-bench, turned out to be useful in transferring knowledge between RE and ST roles and in clarifying misunderstandings between them. By representing the coordination between RE and ST in an artifact map, we used the same heuristics of the dyad structure analysis performed earlier on the alignment methods, leading to the identification of bottlenecks (i.e. synchronization of too many artifacts) and sub-optimizations (i.e. late creation of artifacts) in the interaction between RE and ST. Since REST-bench is very lightweight, it could be integrated into post-mortem procedures that many organizations typically perform after the closure of a development project.

The REST taxonomy provides a novel view on the aligning requirements engineering and software test, based on a rigorous classification scheme and method. It enables the characterization of alignment methods and the assessment of alignment in a development organization. We are continuing our work on utilizing REST-bench as an alignment assessment aid, since the underlying taxonomy and the developed dyad structure properties have a great potential of identifying, characterizing and probing strengths and weaknesses in the alignment between RE and ST in industry, in particular as a sanity check for process improvement initiatives.

## APPENDIX

### *Taxonomy application on alignment methods – Results*

We summarize here the application of the taxonomy on four more cases: Case A [Güldali et al., 2011], Case B [Flammini et al., 2009], Case C [de Santiago Júnior and Vijaykumar, 2012] and Case D [El-Attar and Miller, 2010]. The graphs on the left in Tables 3.4, 3.6, 3.8, and 3.10 illustrate the dyad

Table 3.3: Context of Case A

| Aspect          | Description  |
|-----------------|--|
| Method setting  | 319 requirements, functional and non-functional requirements, requirements-based acceptance testing / test-planning, bespoke RE, natural language requirements                                 |
| Focus           | 4) Main purpose was to improve / affect alignment  |
| Motivation      | Inconsistencies, redundancies and dependencies in requirements documents, representing different viewpoints of a system, lead to erroneous tests, increased test effort and complex test plans |
| Assumptions     | Natural language requirements specified in template-form   |
| Quality targets | Improve test plans and make the test process more efficient  |
| Validation      | Clustered requirements and generated test-plans are reviewed and, if necessary, refined  |

Table 3.4: Dyad structure and characterization of Case A

The diagram illustrates the dyad structure. Three nodes are shown: N1 (Requirement specification), N2 (Requirements analysis), and N3 (Test specification). N1 and N2 are connected by a dyad labeled 'Connection Tool'. N2 and N3 are connected by a dyad labeled 'Transform. Tool'. A vertical dashed line separates N1 from N2.

| ID | Node name                 | Information                        | Owner        |
|----|---------------------------|------------------------------------|--------------|
| N1 | Requirement specification | Nat. lang. specifications          | ∅            |
| N2 | Requirements analysis     | Requirements clusters              | ∅            |
| N3 | Test specification        | Testplan / Ab- abstract test cases | Test manager |

Dyads (2): N1-N2, N2-N3

structures, showing also the mechanism and medium characterizing the link. Note that the empty-set symbol ( $\emptyset$ ) indicates that we could not identify the owner of information in the respective node. Tables 3.3, 3.5, 3.7, and 3.9 summarize the respective method context. Context aspects which could not be identified in the paper are not shown in the tables. In the following paragraphs we describe each case, motivating the identified dyad structures.

CASE A [Güldali et al. \[2011\]](#) present a method and tool support for deriving optimized acceptance test plans from requirements specifications. Table 3.3 summarizes the case context.

We have identified three nodes as illustrated in Table 3.4. The information in N1 is multi-viewpoint requirements specifications, capturing different aspects of the software system and its environment. Since these aspects contain overlaps, the requirements specifications may be redundant. Leveraging on linguistic analysis, similar requirements are clustered (N2) such that a reduced set of test steps and asserts can be derived (N3). The mechanism in dyad N1-N2 is a *connection* since notation and meaning of the information are not changed (individual requirements are mapped into

Table 3.5: Context of Case B

| Aspect          | Description  |
|-----------------|--|
| Method setting  | 1500 abstract and 200,000 executable tests, implementation and verification under safety norms, functional requirements, system testing (instantiation of abstract test cases based on concrete system configurations) |
| Focus           | 4) Main purpose was to improve / affect alignment  |
| Motivation      | Manual instantiation of configuration-specific test cases is a time-consuming activity   |
| Assumptions     | Requires a certain reference software architecture; control system can be abstracted as a FSM  |
| Quality targets | Improve configuration coverage in system tests   |
| Outcome         | Generation of executable test cases is by orders of magnitude more efficient than creating them manually   |

Table 3.6: Dyad structure and characterization of Case B

| ID                      | Node name                 | Information                    | Owner |
|-------------------------|---------------------------|--------------------------------|-------|
| N1                      | Requirement specification | System functional requirements | Ø     |
| N2                      | Test modeling             | Abstract test cases            | Ø     |
| N3                      | Implementation            | System specific configuration  | Ø     |
| N4                      | Test implementation       | Executable test cases          | Ø     |
| Dyads (2): N1-N2, N2-N4 |                           |                                |       |

clusters). In dyad N2-N3 we observe however a *transformation* mechanism since the relationship between requirements is preserved in the derived test plan, expressed in the order of the test steps.

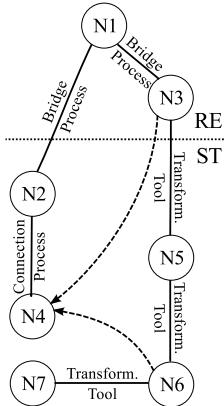
CASE B Flammini et al. [2009] propose a method to automate the verification of computer-based control systems on different configurations. Table 3.5 summarizes the case context.

As shown in Table 3.6, we have identified four nodes. The mechanism in dyads N1-N2 is an *implicit connection* since the description of the process does not reveal how abstract test cases are derived from the functional requirements (the focus of the study is on the instantiation of abstract test cases). For the derivation of executable test cases (N4), the method depends on abstract test cases (N2), and uses the configuration data specific for the system under test (N3). The “use” relationship is shown by the dashed link between N2 and N3. The mechanism in dyad N2-N4 is a *transformation* since the notation of the non-executable test model (abstract test cases) is translated into concrete test cases, preserving the relationships between the abstract test cases by leveraging the information provided in the system specific configuration (N3).

Table 3.7: Context of Case C

| Aspect         | Description   |
|----------------|---|
| Method setting | 175 scenarios derived from requirements, SDLC includes independent V&V, functional requirements, model-based system and acceptance testing, bespoke requirements, natural language requirements |
| Focus          | 5) Intended, main as well sole purpose  |
| Motivation     | Natural language used the most in specifying software requirements and deriving scenarios for system and acceptance tests is challenging and time-consuming                                     |
| Outcome        | Quality of derived executable test cases is comparable to expert's  |

Table 3.8: Dyad structure and characterization of Case C



| ID  | Node name                 | Information               | Owner         |
|---|---------------------------|---------------------------|---------------|
| N1  | Requirement specification | Nat. lang. specifications | Ø             |
| N2  | Scenario definition       | Factors and Levels        | Test designer |
| N3  | Requirement analysis      | Dictionary                | Test designer |
| N4  | Scenario mapping          | SRS to scenario mapping   | Test designer |
| N5  | Test modeling             | Statechart model          | Test designer |
| N6  | Test specification        | Abstract test cases       | Test designer |
| N7  | Test implementation       | Executable test cases     | Test designer |
| Dyads (6): N1-N2, N1-N3, N2-N4, N3-N5, N5-N6, N6-N7 |                           |                           |               |

CASE C de Santiago Júnior and Vijaykumar [2012] describe a methodology to generate scenario-based test cases from natural language requirements. Table 3.7 summarizes the case context.

We have identified seven nodes as illustrated in Table 3.8. The information in N2, factors and levels upon which scenarios are generated, are derived from the software requirements specification (N1). The link mechanism is a *bridge* since in the process, the notation of the information in N2 changes, but the meaning (i.e. the decision whether a certain combination of factors and levels is relevant) has to be maintained by the test designer. The analogous argument applies for the dyad N1-N3, in which the test designer needs to apply his domain knowledge to establish the dictionary (N3). In dyad N2-N4, the test designer establishes a mapping between the generated test scenarios and the related requirements. This information is used (hence the dashed connector) in N3 and N6 to generate statechart models and executable test cases. Since the link in dyad N2-N4 establishes a logical link between requirements and scenarios, we define the mechanism as a *connection*. The dyads N3-N5, N5-N6 and N6-N7 fea-

Table 3.9: Context of Case D

| Aspect          | Description   |
|-----------------|---|
| Method setting  | small-scale example application, functional requirements, acceptance testing, natural language requirements and models              |
| Focus           | 5) Intended, main as well sole purpose  |
| Motivation      | Lack of process that allows analysts to develop acceptance tests from use case models without requiring additional design artifacts |
| Quality targets | Development of comprehensive and effective acceptance tests   |
| Validation      | With robustness diagrams, consistency of use case and domain models can be checked informally                                       |

Table 3.10: Dyad structure and characterization of Case D

| ID | Node name                 | Information                 | Owner        |
|----|---------------------------|-----------------------------|--------------|
| N1 | Requirement specification | Use cases, domain models    | Bus. analyst |
| N2 | Acceptance test design    | High level acceptance tests | Bus. analyst |
| N3 | Requirement analysis      | Robustness diagrams         | Bus. analyst |
| N4 | Acc. test implementation  | Executable acceptance tests | Bus. analyst |

Dyads (3): N1-N2, N1-N3, N2-N4

ture a *transformation* mechanism since relationships between requirements (stemming from the scenario mapping in N4) are preserved in the statechart model (N5), in the abstract test cases (N6) and in the executable test cases (N7).

CASE D El-Attar and Miller [2010] propose a method to derive executable acceptance tests from use case models, domain models and robustness diagrams. Table 3.9 summarizes the case context.

We have identified four nodes as shown in Table 3.10. In dyad N1-N2, the business analyst derives from use case and domain models high level acceptance tests (HLAT) which can be used to evaluate the system manually. We observe a *transformation* mechanism since the procedure for maintaining the relationships between use cases in the corresponding HLATs is well defined. Similarly, robustness diagrams (N3) are constructed to ensure consistency between use cases and domain models. The link in dyad N2-N4 is established by a *connection* mechanism, mapping HLATS with executable test cases and using robustness diagrams (dashed connector N4-N3).

---

## ASSESSING REQUIREMENTS ENGINEERING AND SOFTWARE TEST ALIGNMENT WITH REST-BENCH – FIVE CASE STUDIES

---

**SUMMARY** The development of large, software-intensive systems is a complex undertaking that we generally tackle by a divide and conquer strategy. Companies thereby face the challenge of coordinating individual aspects of software development, in particular between requirements engineering (RE) and software testing (ST). A lack of REST alignment can not only lead to wasted effort but also to defective software. However, before a company can improve the mechanisms of coordination they need to be understood first. With REST-bench we aim at providing an assessment tool that illustrates the coordination in software development projects and identify concrete improvement opportunities. We have developed REST-bench on the sound fundamentals of a taxonomy on REST alignment methods and validated the method in five case studies. Following the principles of technical action research, we collaborated with five companies, applying REST-bench and iteratively improving the method based on the lessons we learned. We applied REST-bench both in Agile and plan-driven environments, in projects lasting from weeks to years, and staffed as large as 1000 employees. The improvement opportunities we identified and the feedback we received indicate that the assessment was effective and efficient. Furthermore, participants confirmed that their understanding on the coordination between RE and ST improved.

### 1 INTRODUCTION

Requirements Engineering (RE) is the discipline of eliciting, analyzing, specifying, validating and managing needs and constraints on a software product [Nuseibeh and Easterbrook, 2000, Bourque and Fairley, 2014]. Software Testing (ST) is the verification that a software product provides expected behaviors, as expressed in requirements [Bertolino, 2007, Bourque

and Fairley, 2014]. As such, RE and ST<sup>1</sup> are intrinsically related and leveraging on this relationship would be beneficial for both disciplines [Graham, 2002]. Nevertheless, relatively few studies have investigated how RE and ST can benefit from an improved alignment (see Section 2).

First steps to a better understanding of the REST alignment phenomenon were undertaken by studying and classifying alignment practices (see Chapter 3). The main contribution of this classification is the definition of an epistemic base [Mokyr, 2005] that can be used to explain how and why REST alignment practices work. In this chapter we use this base in order to provide a practical method, REST-bench, to assess REST alignment. A prerequisite for any improvement is the characterization of the current condition of the phenomenon under study. Based on this agreed state and the definition of goals, changes can be designed and implemented. Post-mortems [Birk et al., 2002] have been proposed to elicit best practices but also issues in the execution of projects, feeding the results into an organizational knowledge repository [Ivarsson and Gorscheck, 2012]. Even though guidelines for executing postmortems exist [Collier et al., 1996, Dingsøyr, 2005], postmortem reviews are seldom held, some suggest for lack of time [Keegan and Turner, 2001, Glass, 2002a], even though their benefits are well reported [Verner and Evansco, 2005].

We designed REST-bench to be lightweight, in terms of resource use, by focusing the assessment effort to the specific issue of requirements engineering and test coordination which is of major interest to organizations developing software intensive systems [Bjarnason et al., 2014a]. In this chapter, we present REST-bench in an example-driven manner, describing data elicitation, data preparation and the collaborative analysis that involves all assessment participants. We illustrate the application of REST-bench in five companies that exhibit diverse characteristics. In all five cases we could identify relevant improvement opportunities. The participants of the assessment judged REST-bench as an efficient and effective mean to assess the coordination between RE and ST.

The remainder of this chapter is structured as follows. We discuss background and related work in Section 2. Section 3 illustrates the research method we followed to validate REST-bench. We introduce the assessment method in Section 4, together with a running example that shows the application of REST-bench and with the improvements we implemented, the collected data, the analysis of the results and the identified improvement potential. In Section 5 we show the results of the remaining four case studies. We answer our initially stated research questions in Section 6 and conclude the chapter in Section 7.

---

<sup>1</sup> We abbreviate requirements engineering and software testing as “RE and ST” or “REST” in the remainder of this chapter.

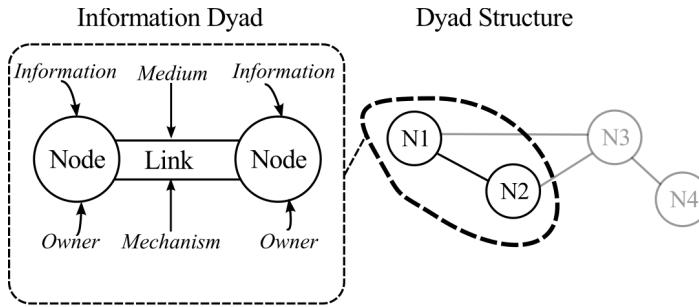


Figure 4.1: Information dyad and dyad structure

## 2 BACKGROUND AND RELATED WORK

### 2.1 REST Alignment

The development of software-intensive systems<sup>2</sup> is a collaborative effort of experts, each contributing a part to the solution [Rus and Lindvall, 2002]. Expertise and capabilities are distributed on different roles, rendering the coordination between people in software projects and development phases essential for success [Kraut and Streeter, 1995]. While research has produced an ample amount of software technologies, models and frameworks to coordinate people, to ease the transition between software development phases and to align the intentions and activities therein, literature is sparse on methods that focus on improving the coordination between requirements engineering and software testing [Bjarnason et al., 2014a]. In earlier work [Unterkalmsteiner et al., 2014a] (see Chapter 3) we defined REST alignment as the *adjustment of RE and ST efforts for coordinated functioning and optimized product development*. The key in this definition is the intuition that RE and ST efforts need to be adjusted together in order to avoid sub-optimization of either one of the two aspects. The concept of REST alignment goes thereby beyond traceability, which can however be one mean to achieve alignment [Unterkalmsteiner et al., 2014a].

### 2.2 The REST Taxonomy

To characterize means aimed at achieving REST alignment, we developed a taxonomy that uses the information dyad as building block to describe alignment methods [Unterkalmsteiner et al., 2014a] (see Chapter 3). Figure 4.1 illustrates the components of an information dyad: two nodes, char-

<sup>2</sup> A software-intensive system is “any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole” [ISO/IEC, 2007].

Table 4.1: Dyad structure properties

---

|                       |  |
|-----------------------|--|
| <i>P</i> <sub>1</sub> | Number of nodes – links between nodes need to be maintained over time. Hence, the total number of nodes allows one to reason on complexity and effort to maintain REST alignment.  |
| <i>P</i> <sub>2</sub> | Branches – a branch exists, if a node acts as a source or sink for more than one node. Branches may reduce local complexity, require however also means to synchronize and merge information.  |
| <i>P</i> <sub>3</sub> | Intermediate nodes – characterized by information that belongs to the design/analysis or implementation phases.  |
| <i>P</i> <sub>4</sub> | RE and ST node proportion – assuming that a node is associated with a certain cost (e.g. establishing/maintaining the information therein and links in between), it is of interest to know the node distribution among the RE and ST phases. |
| <i>P</i> <sub>5</sub> | Links – the linking mechanism between phases determines how changes of information are propagated.   |
| <i>P</i> <sub>6</sub> | Scope – allows one to reason upon the interface between RE and ST and other phases.  |

---

acterized by the information they contain and their owner, are connected through a link, described by a linking medium and mechanism. Information dyads form a structure which in turn may exhibit certain properties, listed in Table 4.1. These properties can be used to compare REST alignment methods, analyzing their complexity and scope. More details on how these building blocks of the REST taxonomy were derived are described in Chapter 3 where we also piloted the assessment method which is the focus of this chapter. REST-bench uses dyad structures and their properties to assess alignment as described in Section 4.

### 2.3 Related Work

The challenges of coordinating development teams operating at different sites have been described by Herbsleb and Grinter [1999]. They point out the boundaries of explicit coordination mechanisms such as plans, interface specifications and process descriptions, but also the lack of informal coordination opportunities in geographically distributed sites, leading to misunderstandings and increase in cycle time for fixing issues. Following this line of thought, Herbsleb and Mockus [2003] developed an empirical theory of coordination (ETC) for software engineering, based on decision and constraint networks, and later applied to identify coordination requirements among software developers that can be used to improve the design of collaboration tools [Cataldo et al., 2006]. Along a similar line, Ko et al.

[2007] observed in a field study what information developers seek in their day-to-day work, which sources they use and what the reasons are for not being able to gather the needed information, calling for innovation in tools, processes and notations. REST-bench shares with these studies the insight that satisfying information needs, timely and as exhaustive as possible, is key to successful software development in teams.

In order to represent information flow in requirements engineering activities, Schneider et al. [2008] developed a notation that can be used to model both formal and informal communication. A benefit of the flow notation is that it can be used to describe the officially required and the actually executed process, showing differences and instances where improvements can be implemented [Stapel et al., 2007]. Furthermore, FLOW mapping has been used to plan and manage communication in distributed teams, requires however a considerable amount of manual work to keep the maps up-to-date with continuously changing communication patterns [Stapel et al., 2011]. REST-bench shares with FLOW the idea to represent information and connections in a diagram that can be discussed in collaboration with practitioners to identify improvement opportunities. On the other hand, REST-bench provides a concrete assessment process, differentiates between data collected from the different roles, and provides heuristics that can be used to analyze the collected data and to generate analysis points that may lead to improvement suggestions.

The idea of using project retrospectives as sources for identifying improvements is not new [Collier et al., 1996]. However, they are seldom conducted due to the fast paced nature of software projects where teams are split up and reassigned to new tasks [Glass, 2002a]. Therefore, collecting information timely after the conclusion of a project seems more likely to identify improvement potential. In order to increase the data accuracy, Bjarnason et al. [2012] propose project timelines that are prepared in advance. Depending on the particular analysis goals, certain aspects of the collected data are visualized in a timeline, allowing the postmortem participants to recall the illustrated events. However, since the method can be used for any generic improvement goal, it does not provide any prompting questions or aids that could facilitate the analysis of the collected data. REST-bench relies also on the collection of data from a specific past project, eliciting practitioners' experience on how information is used and created. The focus on a particular goal (coordination of RE and ST) and a data analysis guided by a set of predefined questions, sets REST-bench apart from traditional project retrospectives.

### 3 RESEARCH METHOD

Our overall research approach is oriented towards design research [Hevner et al., 2004] which provides a concrete framework for implementing the dynamic validation phase in the technology transfer model proposed by Gorscheck et al. [2006]. Our research method is best described as technical action research [Wieringa, 2014a, Ch. 19], i.e. we aim at improving and validating the fitness of purpose of an artifact by applying it in a real-world environment [Wieringa, 2014b]. In particular, we want to answer the following research questions:

- RQ1: To what extent are the dyad structures from the REST taxonomy useful to elicit improvement opportunities?
- RQ2: To what extent is REST-bench useful in Agile and plan-driven environments?
- RQ3: Is REST-bench usable?

In our previous work, we characterized REST alignment methods by means of information dyads [Unterkalmsteiner et al., 2014a] (see Chapter 3). Furthermore, we piloted the idea of using dyad structures as a mean to drive REST alignment assessment. With RQ1 we aim to validate this idea by applying the REST-bench method in a series of case studies. When we planned the validation, one of our concerns was the methods' reliance on documentation as a proxy to determine REST alignment. Therefore, we questioned whether we can apply REST-bench at all in an Agile environment. We address this concern in RQ2, where a plan-driven environment means that the development teams work in a traditional, document driven manner [Petersen and Wohlin, 2010]. In order to be adopted by practitioners, a method should also be usable. We analyze the usability of REST-bench from the perspective of the analyst, i.e. the researcher who applied the method, as well as through the practitioners' feedback who participated in the study.

#### 3.1 Project Characteristics

We applied REST-bench in five companies located in Sweden: Ericsson and Telenor in Karlskrona, ST Ericsson in Lund, CompuGroup Medical and Volvo Cars in Gothenburg. All companies were approached based on personal contacts, providing them an executive summary of the goals, and expected cost and benefits of REST-bench. We did not preclude any particular company or project type in the selection. In the remainder of this chapter we anonymized project characteristics, collected data and results by referring to Company A, B, C, D, and E. Table 4.2 illustrates the characteristics of the projects where we applied REST-bench. Note that Company

Table 4.2: Project characteristics of the participating companies

| Company                       | A                                      | B                  | C                             | D           | E           |
|-------------------------------|--|--------------------|-------------------------------|-------------|-------------|
| Project duration              | 12 months                              | 2 months / release | 36 months                     | 48 months   | 7 months    |
| Staff                         | 150                                    | 20                 | 9                             | 1000        | 20          |
| Software development approach | Agile (in transition from plan-driven) | Agile              | Agile embedded in plan-driven | plan-driven | plan-driven |
| Requirements #                | 350                                    | 5-20               | 300                           | 2000        | 50          |
| Test-case #                   | 700                                    | Not stated         | 300                           | 500         | 24          |
| Assessment date               | 2012/06                                | 2013/03            | 2013/03                       | 2013/04     | 2013/12     |

A was, at the time of the assessment, in a transition from a plan-driven to an Agile software development approach. The team in Company C was working in an Agile manner while still being embedded in a plan-driven process. We selected the particular projects based on the criteria defined by REST-bench, described in Section 4.1.

### 3.2 Data Collection

In order to answer the previously stated research questions, we dedicatedly collected data but also used data that was produced while applying REST-bench during each assessment at the corresponding company. For example, a part of the answer to RQ1 stems from the observation which dyad structure properties (see Table 4.1) actually triggered seeding questions for the assessment. Furthermore, we collected the spent effort in order to provide input to the question whether REST-bench is usable (RQ3). However, the main data source was the dedicated post assessment questionnaire that we distributed to the participants. Table 4.5 shows the 10 questions. The first four questions are open ended, addressing the overall experience of using REST-bench. The remaining questions required an answer on a 5 point Likert scale (strongly agree, agree, neutral, disagree, strongly disagree), while explanatory feedback was still possible. Out of the 13 participants in total, 10 returned the questionnaire.

### 3.3 Limitations

A major threat to validity in any technical action research is the involvement of the researcher in the application of the to be validated artifact [Wieringa and Morali, 2012]. The question whether the validation results depend solely on the ability of the researcher to use the artifact cannot be answered conclusively. The first author performed all steps in REST-bench. An alternative would have been to train practitioners and to let them apply REST-bench on their own, observing the application and collecting data of spent effort, usefulness and usability. However, this would have led to a considerably higher assessment cost per company. Nevertheless, the validation did not solely depend on the researcher since practitioners were involved in the REST-bench application process, influencing thereby considerably the results.

We applied REST-bench in projects with a diverse set of characteristics (see Table 4.2). There is no feature or constraint in REST-bench that could conceivably exclude certain classes of projects or companies, even though very small teams that work co-located and do not require coordination with other teams might not benefit from an assessment with REST-bench.

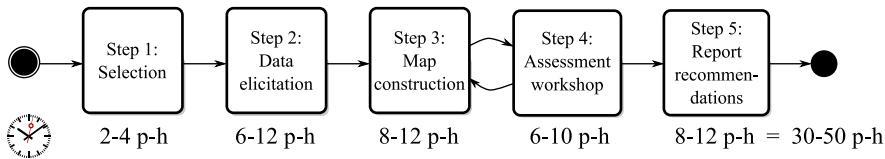


Figure 4.2: The steps in the REST-bench method and budgeted effort in person-hours (p-h)

#### 4 REST-BENCH

REST-bench follows the macro procedure of lightweight software process assessment [Pettersson et al., 2008], however it has a focused improvement goal (coordination between RE and ST) and utilizes elicitation and analysis practices tailored to this goal. This focus leads to a low investment cost, typically between 30 and 50 person-hours (p-h), depending on how many people are involved in the assessment (typically 3 to 5). We have a project view to make the assessment concrete, and avoid people give us the best or worst cases picked together from any part at any time in the company. This project focused assessment has been used before with success [Svahnberg et al., 2015].

The aim of REST-bench is to assess the state of REST alignment [Unterkalmsteiner et al., 2014a], eventually initiating changes to improve that alignment. A critical enabler for any kind of change in an organization is support by senior management [Dybå, 2005, Niazi et al., 2006, Wohlin et al., 2012]. Therefore, we regard a champion in the organization that can provide resources and support the dissemination of the assessment results an essential pre-requisite before embarking in a REST-bench assessment.

We performed a REST-bench assessment at five case companies, and used the experiences to refine REST-bench. The first four cases (A-D) are illustrated in chronological order in Section 5, where we show the results of the assessment and the adaptions we made to the method. In this section we use the fifth case (E) as a running example to illustrate the individual steps of REST-bench. In all five case illustrations, the author of this chapter acted as moderator and analyst during the assessment.

REST-bench is interview-driven and begins therefore with the selection of interviewees (Step 1 in Figure 4.2). After data elicitation (Step 2), the analyst creates an artifact map (Step 3). In Step 4, the analyst meets with the interviewees to collaboratively identify improvement opportunities, using the artifact map as input. The results of this assessment workshop are collected in a report (Step 5). In subsections 4.1 - 4.5 we exemplify the steps shown in Figure 4.2, providing effort estimation and a compilation of best practices for the application of REST-bench.

#### 4.1 Step 1 – Selection

REST-bench has an interview-driven data elicitation process. Therefore, the selection of the particular interviewees is pivotal for the assessment, not unlike most process assessment methods such as e.g. CMMI SCAMPI [SEI, 2006] or SPICE [Rout et al., 2007]. A local champion in the organization, supporting the initiative, can be a great accelerator in identifying interview candidates. In order to maintain REST-bench’s lightweight and focused nature, the roles of the 2-4 interviewees should pertain to the requirements domain, e.g. requirements engineers, business analysts, product managers, and the testing domain, e.g. test and quality assurance engineers. We recommend that the selection process considers the following characteristics:

1. Work experience in the company and in the particular role. A candidate that knows “how things work” and has progressed in the same company to a senior position is preferable over a qualified, but new hire.
2. The candidate from the RE and ST domain must have collaborated on the same project, preferably in more than one instance.
3. The project has recently been closed or is in a late stage.

These characteristics allow us to elicit data that reflect how engineers actually work as opposed to how they ought to work according to a process prescribed by a company policy. Choosing a project on which all interviewees collaborated allows us to conduct episodic interviews which facilitates the collection of “everyday knowledge about objects or processes” [Bauer and Gaskell, 2000, p. 85]. In episodic interviews, we try use actual events that are connected to actions, experiences and consequences, to elicit relevant information from project participants. A project that is in its early stage can not set the context as this everyday knowledge (episodes) has still to be acquired. We detail data elicitation in Section 4.2.

In some organizations, the notion of a “project” is not clear cut. For example, one case company applies SCRUM [Schwaber, 1997] with three week sprints to implement customer requirements, releasing a new version of the product every two sprints. In this situation we chose the last sprint as “project” scope for the interviews. The relevant decision here is to agree on a specific context such that all interviewees can relate to it.

When selecting the interviewees it is important to keep the main goal of the assessment in mind: identify improvement opportunities for the coordination between requirements engineering and testing aspects of software development. Optimal candidates are employees in a senior position, being however involved in the day-to-day practice in their respective area. This leads usually to very fruitful analysis during the collaborative issue identification (Section 4.4).

## *Effort and Best Practices*

The budgeted effort for this step is 2-4 person-hours, where the majority of the expense is on the organization, identifying the interview candidates. The analyst can support this by providing detailed selection criteria for projects and interviewees, described in this section. We recommend to schedule all meetings in advance (interviews, workshop) such that the overall assessment procedure is not extended to a long time period. Interviews should be held on one day, increasing the efficiency of data collection. The assessment workshop should be scheduled between 4 and 10 business days after the interviews, allowing time to analyze the collected data and preventing a loss of context by waiting too long after data collection.

### *Example – Step 1*

Company E outsources the implementation and verification of parts of their product to an external supplier. In this scenario, the coordination between RE (Company E) and ST (supplier) is particularly challenging since company, country and time-zone borders are crossed. The selection of interviewee candidates was supported by the company's responsible for process improvement. We organized a seminar for employees of Company E, presenting the goals and example applications of REST-bench. This pro-activeness raised interest in the method and helped us to identify the project and interviewees for the assessment. We chose an Analyst Lead and a Business Process Expert/Acceptance Tester for the RE perspective, both working for Company E. For the ST perspective, we chose an Acceptance Test Manager working at Company E and the System Test Manager working for the supplier. The chosen project had a duration of 7 months involving a total staff of 20 employees (both Company E and supplier).

### *4.2 Step 2 – Data Elicitation*

We argued in our earlier work on requirements engineering and software testing alignment methods [Unterkalmsteiner et al., 2014a] (see Chapter 3) that the means of connecting and using information is central to any software development effort. In order to operationalize this concept in a data collection procedure, we elicit which artifacts requirements and test engineers use and create in their daily work. We chose artifacts as a tangible proxy for information which can be retrieved during an interview. Furthermore, every software organization produces some set of artifacts, independently how agile or lean the organization is. An artifact can be any kind of digital or analog document produced by an employee. In our interview

Table 4.3: Example artifacts

---

Powerpoint presentations, spreadsheets, text documents, specification / use case / user story stored in requirements management tool, acceptance / integration, system, unit test case stored in test management tool, UML / entity-relationship diagrams, emails, meeting notes, yellow sticky notes

---

guide we exemplify what we mean by artifacts (see Table 4.3 for a list of examples).

Agreeing on the concept of artifacts is key to the episodic interview technique as it sets the scope of what data is elicited. Therefore, Phase 1<sup>3</sup> requires the interviewee to list all artifacts he/she has used or created in his/her daily work. The interviewee should follow the timeline of the project under investigation, recalling his/her involvement during the different project phases that required use or creation of any artifact. In Phase 2 of the interview we elicit the following data on each of the artifacts:

- Purpose: content of the artifact and reason for its creation
- Creator: role of the person who creates the artifact, in which phase of the project
- User: role of the person using the artifact, in which phase of the project
- Modifier: role of the person changing the artifact, in which phase of the project
- Link or Mapping: a link is a uni-directional connection from one artifact to another; a mapping is a bi-directional connection between information contained in two artifacts.
- Use: a reference to any other artifact that is used as input to create this artifact

We collect this data by compiling a simple template (see Figure 4.3). It is common that the list elicited in Phase 1 is incomplete and is extended while detailing the artifact information in Phase 2 of the interview.

In order to reduce mutual influence during data collection, we interview requirements and test engineers separately. We also recommend to audio record the interviews (provided the interviewees give consent). With the episodic interview technique, we don't elicit data on artifacts in a vacuum, but enrich the information by relating it to the studied project and what has been experienced in the use and creation of artifacts. The analyst can then, while constructing the artifact map (see Section 4.3), develop a bet-

---

<sup>3</sup> In Phase 0, we ask a set of introductory questions to elicit context information, such as name of the project, project duration, staff size, applied software development method, number of system requirements and test cases.

ter understanding of the coordination between RE and ST, and prepare targeted analysis points for the assessment workshop.

### *Effort and Best Practices*

The budgeted effort for this step is 6-12 person-hours. We allocate 1.5 hours per interview, which translates into 6 person-hours for 1 analyst and 2 interviewees as we interview RE and ST separately. Adding two interviewees doubles the effort to 12 person-hours.

Interviewees should have access to the project documentation during the data elicitation. This allows them not only to exemplify the artifacts they mention, but also to provide more accurate information on how artifacts are linked and related to each other.

### *Example – Step 2*

The three interviews at Company E were performed on two days. The ST representative from the supplier company was interviewed by phone, however not audio recorded upon request by the interviewee. Figure 4.3 shows an excerpt of the simple elicitation form used to record data. We experienced that notes are sufficient to record facts about artifacts, eliciting them in a structured fashion. However, episodic information on their use or misuse is complex and requires audio recordings that can be analyzed offline. When interviewees refer to events in the project, the analyst should focus on understanding these occurrences, in order to pose follow-up questions, rather than spending his attention on recording them manually.

### *4.3 Step 3 – Map Construction*

An artifact map is an easy to understand, graphical representation of the data collected during the interviews. The visualization serves mainly three purposes:

1. Illustrating commonalities and differences of the RE and ST perspective on use and creation of artifacts.
2. Generating input for the collaborative issue identification.
3. Communicating and displaying the artifact topology to employees not involved in the assessment.

Figure 4.4 illustrates the components of an artifact map. A rectangular box represents an artifact, a sphere with an identifier indicates a creator (on the top-left of the box) or an user (on the top-right of the box) of an artifact. “Linked-to” relationships are indicated by solid lines with a single arrow, “mapped-to” relationships by solid lines with double arrows, and “used-to-create” relationships are shown by dashed lines with a single arrow. The

| ID | Purpose  | Creator / When (activity)  | User / When (activity) / Purpose  | Who |
|----|--|--|---|-----|
| ①  | Requester info<br>purpose of<br>project<br>Gains/Loses             | Business Manager<br>(Main requester)<br>Before project<br>starts                   | Business analyst/<br>Steering group of<br>product House                       | B   |
| ②  | Want all needs<br>from different<br>perspectives                   | BM (responsible)   | BA/Solution Lead -<br>Business Analyst<br>early (also early start)<br>startup | K   |
| ③  | Understanding<br>of supported<br>Business/Real<br>data for testing | Business Manager / Oracle<br>Product Manager<br>Throughout<br>the project lifetime | BA / Acceptance Tester<br>early late in<br>project                            | C   |
| ④  | Details of<br>product/<br>calculator<br>of services                | Oracle<br>during the<br>project  | Acceptance Tester /<br>→ test data  | O   |

Figure 4.3: Artifact elicitation example

color coding indicates whether the data stems from RE (orange), ST (blue) or from both roles (green). The map in Figure 4.4 reads as follows: *RE and ST agree that Artifact A is created by R1 and used by R2, even though ST adds R4 as an user. RE states that Artifact C is created by R3 and is linked to Artifact A. ST states that Artifact B is created by R4, using Artifact A as input, and information in Artifact B is mapped to information in Artifact A.*

This example illustrates a rather common situation where ST states that a role (R4) uses information from one artifact (A) to create another artifact (B), whereas RE is not aware of this information need. This leads to an analysis point that needs to be addressed in the collaborative assessment workshop. One outcome could be that missing to mention Artifact B and R4 was an oversight by RE. Then the map is simply updated. Another potential outcome is that RE is indeed unaware of the information need by R4. This misalignment could then be analyzed in more depth during the assessment workshop, identifying potential causes but also consequences. For example, if Artifact A is seldom updated, R4 might use outdated information to create Artifact B. The solution could be to maintain Artifact

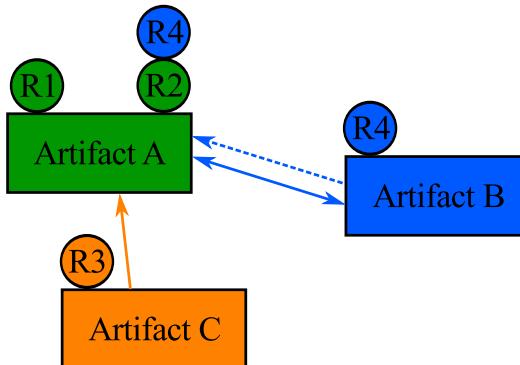


Figure 4.4: Artifact map – Illustration of components

A during the project. Another likely solution: R<sub>4</sub> should use Artifact C which would otherwise have no use (no user was defined for it during the elicitation).

Table 4.4 lists questions that the analyst can use to identify analysis points for the assessment workshop. The questions in set *Po* focus on finding causes of disagreement between RE and ST, which could be a misunderstanding in data elicitation, but also a genuine divergence that potentially causes misalignment. The questions in sets *P1*, *P2*, *P5* and *P6* are mapped to the information dyad structure properties identified in our previous work [Unterkalmsteiner et al., 2014a] (see Chapter 3). Note that these questions are meant as an inspiration for the analyst and do not apply to every artifact map. While applying the REST-bench method, we improved the formulation and extended the set of questions.

An artifact map encodes only a subset of the information elicited during the interviews. It is the task of the analyst to use the remaining data together with the collected episodic information to create analysis points for the collaborative assessment workshop. For example, the data elicitation may reveal that an artifact is regularly updated during a project. However, there is no mechanism to propagate changes to artifacts or roles that use this modified information, leading potentially to misalignment. Episodic information, such as an event where an ST had to redesign a test suite due to outdated requirements specifications can be supporting evidence for a pattern observed in the artifact map.

#### *Effort and Best Practices*

The budgeted effort for this step is 8–12 person-hours. Some effort (40%) should be spent to represent the artifact map graphically. Representing the artifact map in a clear, easy to understand graph is crucial for the assessment workshop since the participants do not have much time to familiarize

Table 4.4: Seeding questions to prepare the REST-bench assessment workshop

|   |
|---|
| <i>Po – What is the source of disagreement between RE and ST on...</i>  |
| 1. ... the existence of an artifact?  |
| 2. ... the creator/user of an artifact?   |
| 3. ... who changes when an artifact?  |
| 4. ... "linked-to"/"mapped-to" relationships between artifacts?   |
| 5. ... linking mechanism between artifacts?   |
| 6. ... "used-to-create" relationships between artifacts?  |
| <i>P1 – Number of artifacts</i>   |
| 6. Is there an information need that was not fulfilled by the used artifacts?   |
| 7. If a new artifact is added, how would that impact other artifacts in terms of maintaining information consistent?  |
| 8. Given that artifact A doesn't have any user OR is only used by role R, could the information in artifact A be merged into artifact B?  |
| <i>P2 – Artifact relationships</i>  |
| 9. How is the information in artifact A kept consistent with the information in artifact B, in the case C changes (C has two "linked-to" OR "used-to-create" relationships to sibling artifacts A and B)? |
| 10. If inconsistencies between artifacts A and B arise, how does that impact the users of those artifacts and their work?   |
| 11. What is the purpose of artifacts that are not related to any other artifact?  |
| 12. Do the creators of artifact A deliver timely, i.e. can the information actually be accessed in ST when needed?  |
| <i>P5 – Artifact and role changes</i>   |
| 13. Does inconsistency of information among artifacts affect the work in: RE, ST, the interface between both?   |
| 14. In case requirements change, by whom/how/when are these changes propagated to linked artifacts?   |
| 15. How does staff turnover affect the quality of requirements and derivative artifacts?  |
| <i>P6 – Artifact and role scope</i>   |
| 16. Would involvement of RE/ST in creating artifact A improve the alignment?  |
| 17. How is consistency between input from non-RE/ST artifacts and RE/ST artifacts maintained over time?   |

with complicated notation. In the cases studies, we used an all-purpose diagramming tool [yWorks, 2014], however have since then developed a dedicated tool<sup>4</sup> that supports the analyst in creating the artifact map. The remainder of the effort should be spent on preparing a list of questions for the assessment workshop, using the prepared artifact map and the audio recordings of the interviews. Note that the audio recordings are not meant to be transcribed, but serve as source for details that were not captured yet in the map and to verify the created map. The list of questions should be separated into a clarifying part, addressing potential misunderstandings from the interviews, and an analysis part, seeded by the questions from Table 4.4.

### *Example – Step 3*

Figure 4.5 shows the artifact map elicited at Company E. Note that for reasons of presentation clarity, we omit in this figure many details such as artifact names and creators/users and highlight only important analysis points in panels A and B (discussed in Section 4.4). The color coding (orange for RE, blue for ST and green for artifacts mentioned by both perspectives) illustrates the degree of agreement on the creation and use of artifacts. The commonly identified artifacts represent thereby the interface between RE and ST. Inconsistencies in the map were picked up as entry points for analysis during the assessment workshop, for example:

- Even though both RE and ST agree on Solution Definition and Development Requirements Specification (see Figure 4.5 and panel A therein), RE states that there is an explicit mapping between individual requirements and solution descriptions while ST claims there is no linking at all.
- The Test Strategy artifact was mentioned by the ST at Company E, but not by the ST at the supplier (see Figure 4.5 and panel B therein).
- A disagreement between RE of Company E and ST of the supplier emerged on the change frequency and time of the Development Requirements Specification (not visible in the artifact map).

These divergences between the RE and ST perspective on the artifact map are a result of the seeding questions Po in Table 4.4 and are complemented with further questions originating from this seeding set, as shown in the next step.

### *4.4 Step 4 – Assessment Workshop*

The REST-bench assessment workshop is led by the analyst, introducing to all interviewees the process of the collaborative issue identification:

---

<sup>4</sup> A prototype is available at <http://lmsteiner.com/restbench>

1. Presentation and explanation of the artifact map
2. Error identification by interviewees
3. Clarification questions by analyst
4. Collaborative map analysis
5. Summary and wrap up

The analyst brings two printouts of the artifact map to the workshop, one for himself to note any corrections and one for the participants, explaining the notation and content of the artifact map. He should also point out immediately that the information in the map stems from the interviews, setting the context to the particular project that was selected for the data elicitation. The interviewees should spend 5–10 minutes with the map and try to identify any unclear artifacts, roles or incorrect relationships between artifacts. The analyst clarifies any uncertainties that emerged in the map construction<sup>5</sup>. The main task on which most of the allocated time should be spent on is to answer the questions prepared by the analyst, based on Table 4.4.

As for the initial data elicitation, we recommend to audio record the assessment workshop, while the analyst should take notes of the major identified issues. An issue might be related to the process (or not followed process), structure, content or distribution of artifacts, coordination between roles using artifacts, documentation infrastructure or lack thereof. It is important that all participants agree on an issue and the implications of it. The analyst should elicit evidence of these implications, e.g. in form of events during the project under discussion, from the workshop participants.

#### *Effort and Best Practices*

The budgeted effort for this step is 6–10 person-hours. A realistic time-frame for the REST-bench assessment workshop execution is 2 hours.

Even though the analyst initiates and drives the assessment workshop with prepared analysis points and questions, improvement opportunities are most likely to be acted upon when they emerge from the company employees. Therefore, the main goal of the analyst is to provoke an exchange between RE and ST, supported by the artifact map as a mean of communication, avoiding however conflicts and steer the discussion into a constructive direction. During the assessment workshop it is also important to maintain the context of the studied project, even though extrapolation to other projects is possible in order to understand the impact of an identified improvement opportunity.

---

<sup>5</sup> This is why map construction and collaborative issue identification in Figure 4.2 are shown as iterative steps.

### Example – Step 4

Looking at Panel A in Figure 4.5, we can observe that the Development Requirements Specification (i.e. the requirements therein) is mapped to the central artifacts in acceptance testing, system and integration testing, and development (Implementation Specification and Solution Definition). This mapping allows the company to assess the coverage of requirements both in terms of testing and implementation, enabling the monitoring of both development and testing progress. Similarly, it can be argued that a mapping between individual business requirements (e.g. from the Business Requirements List or the Idea Document) and the Development Requirements Specification would be important. Indeed, RE confirms that it is not always clear which business requirements are already in production.

Such a mapping is however difficult to achieve, since the Business Requirements List is not a standardized document and does also contain information not relevant for the requirements analysis. RE states that the naming of use cases (part of the Development Requirements Specification) should describe the corresponding business requirement, creating an implicit mapping. This is however not trivial and requires a lot of thought, as the use cases must illustrate the business process such that the use cases can be used “as-is” documentation for other projects. Different roles have different requirements on the use cases and their names, making them not necessarily ideal for determining coverage of business requirements. Therefore, it might be better to pursue an explicit mapping, requiring however a formalization of the business requirements.

Looking at the issue from the perspective of the customer (internally, this is a Business Manager who requests a set of features), RE states that it is sometimes difficult to get the Development Requirements Specification reviewed. It is the customers’ responsibility to look into the Development Requirements Specification and understand what they have agreed upon and verify the coverage of the business requirements. Even though it worked well in this project, it is still challenging in general to get customers to read the Development Requirements Specification and confirm that the business requirements are covered.

Looking at Panel A in Figure 4.5, we can observe a second issue: on the acceptance level, test cases are created by using information from the Product Specification, Product Details/Configuration, Development Requirements Specification and the Solution Definition. Business requirements, represented in the Business Requirements List and the Idea Document are not used at all. Given the lack of mapping between business requirements and solution requirements resulting in a potentially unknown business requirements coverage, using business requirements as input for creating acceptance test cases seems therefore to be a good idea.

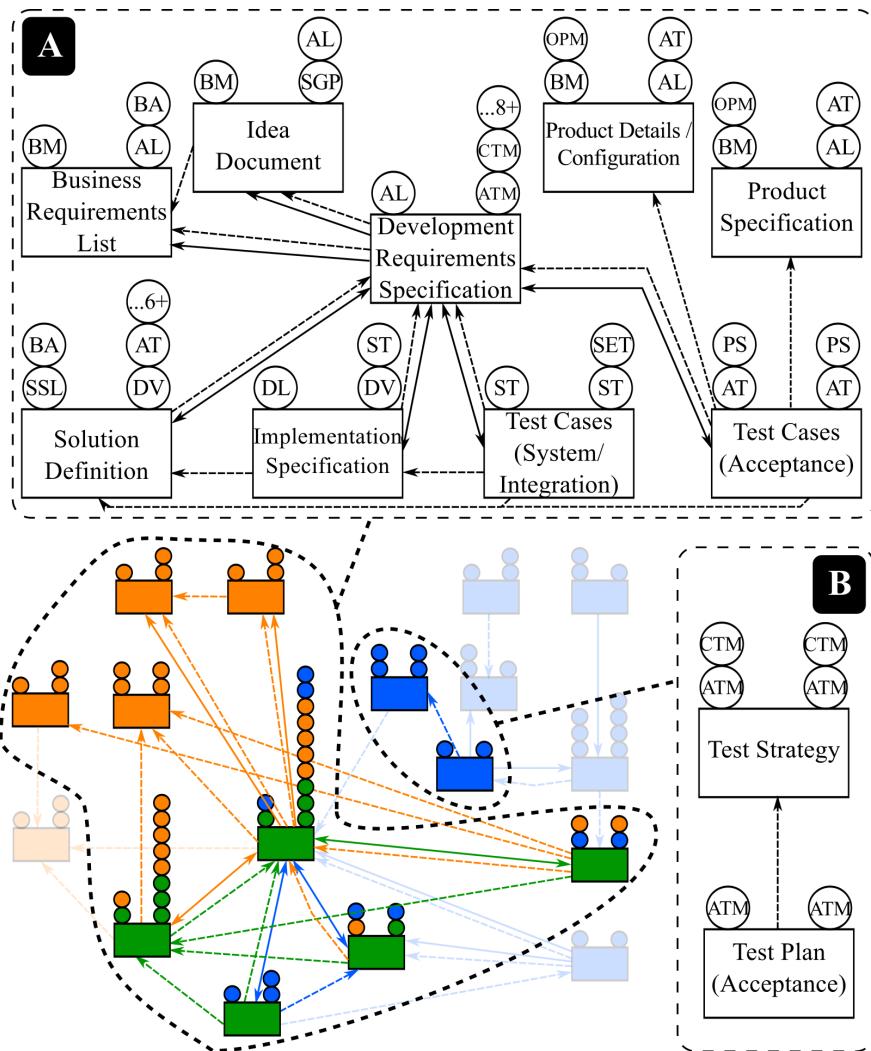


Figure 4.5: Artifact map of Case E

This issue was controversially discussed during the assessment workshop. On one hand, it should not be required to use the business requirements directly as input, since all business requirements selected for the project should be represented in the Development Requirements Specification and the use cases therein. On the other hand, the Development Requirements Specification and the Solution Definition in particular are documents of the solution space, tending to describe the designed solution rather than the problem. The purpose of acceptance tests is to verify that the problem has been addressed by the solution whereas the purpose of system/integration tests is to verify that the solution implements the requirements correctly. A potential consequence of broadening the purpose of acceptance tests is that verification is duplicated, e.g. covering aspects that have been already covered in system tests. Indeed, RE confirms that this can happen whereas ST (at Company E) states that this is not the case in general.

Less controversial was the observation on Panel B in Figure 4.5. Both RE and ST agree that input from the RE perspective on ST plans and strategies would be beneficial. A review of these artifacts by RE would validate that the tested functionality is the actually required functionality.

#### *4.5 Step 5 – Report Recommendations*

The purpose of this step is to summarize the findings from the workshop and to communicate them to a wider audience. For that reason, the report should be concise while still providing enough context in order to be useful for employees that did not participate in the assessment or in the studied project.

##### *Effort and Best Practices*

The budgeted effort for this step is 8–12 person-hours. We recommend to provide a short introduction of the assessments purpose and scope. Then, the artifact map, based on the data collected in the interviews, should be presented, highlighting inconsistencies and analysis points relevant for the assessment workshop. The workshop summary should answer these analysis points, summarize the identified improvement opportunities, pointing to evidence in the updated artifact map, and eventually conclude with a set of recommendations. The report should be send to the study participants for review before it is communicated to a wider audience, allowing for corrections by the participants.

##### *Example – Step 5*

In summary, three areas for improvement were discovered:

- Business requirements gap: the Development Requirements Specification needs to be reviewed in order to verify coverage of business requirements. The effectiveness of the review depends on the skill/capability of the customer in understanding the Development Requirements Specification. These skills can be strengthened. On the other hand, an explicit mapping from Development Requirements Specification to business requirements would reduce the gap too. However, such a solution requires a formalization of the Business Requirements List.
- Acceptance and System/Integration Test alignment: there are indications that these test-levels have at least a certain overlap in what they actually verify. As long as this overlap is intentionally created, the additional effort can be controlled, otherwise this is an area with efficiency improvement potential.
- Involvement of the RE perspective, e.g. involving RE in Test Strategy and Test Plan reviews would help in verifying the acceptance test coverage of the business requirements.

## 5 CASE STUDIES

In this section we present the four remaining cases where we applied REST-bench, chronologically such that improvements in the method follow logically from the described cases. Each subsection introduces first the case company, summarizes the assessment results and discusses the impact the lessons learned had on REST-bench.

### 5.1 *Company A*

We selected a system manager with 12 years experience in his current role as RE representative. For the ST representative we chose a verification engineer with 14 years experience. The project in which the two engineers collaborated completed in autumn 2011, while we performed the assessment in June 2012. The project staff consisted of 150 engineers, split up into seven teams. The system requirements consisted of approximately 350 user stories, whereas the system test cases amounted to 700, of which 550 were automated.

#### 5.1.1 *Assessment Results*

Figure 4.6 shows the artifact map used during the REST-bench assessment workshop. For clarity, we highlight only those details of the map (Panel A and B) relevant for the identified improvement opportunities.

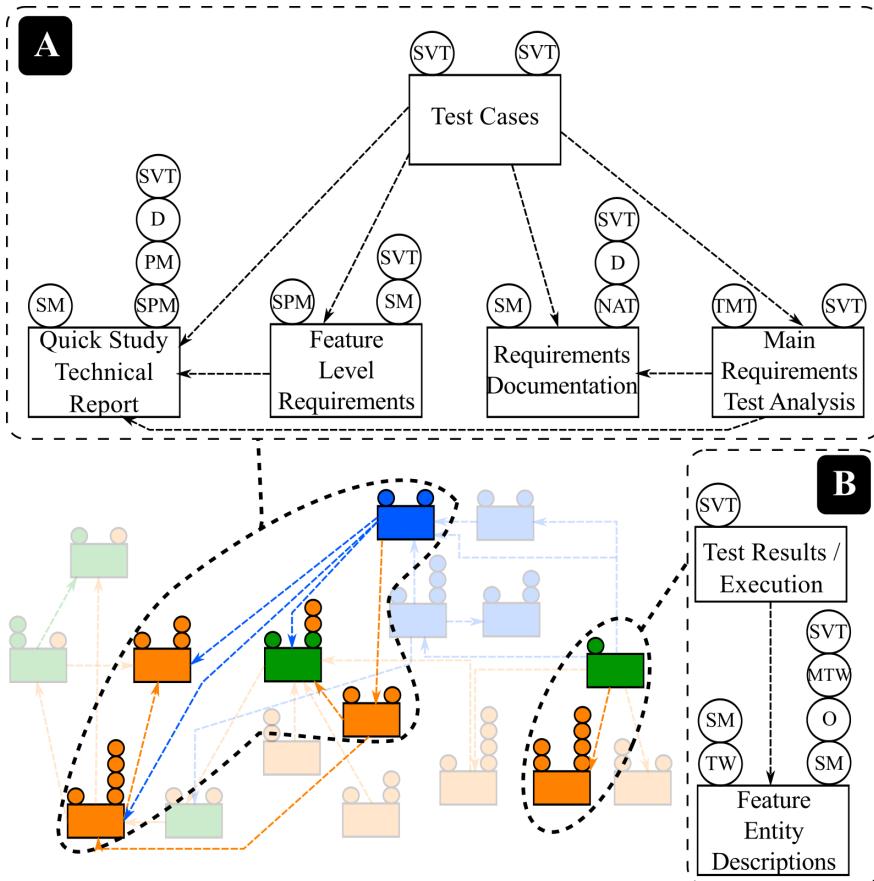


Figure 4.6: Artifact map of Case A

Panel A in Figure 4.6 illustrates that the Software Verification Team (SVT) uses four artifacts as input to create the Test Cases. The Requirements Documentation is the main source, complemented by the other artifacts. During the assessment workshop the question arose whether inconsistencies in these documents, caused for instance by requirements changes during the design or implementation, affect ST. According to RE, changes in user stories (part of the Requirements Documentation) are propagated to the SVT, which is invited to the presentations where changes are discussed. In the context of this particular project, RE stated that parts of the solution were redesigned, leading to consequences which were not dealt with. This was confirmed by ST, asserting that there were inconsistencies between Test Cases and Requirements Documentation due to a too early test analysis.

The Requirements Documentation artifact was elicited from both interviewees and represents therefore an important interface between RE and

ST. According to ST, there is however no mapping between Requirements Documentation and Test Cases at the Integration Test level. This is due to the difficulty to keep this mapping up-to-date; early attempts with spreadsheets failed and importing the required information into the test management tool is labor intensive. According to RE, this lack of mapping may lead to lower test coverage of the requirements. ST states that the lack of traceability is to some extent compensated by the Technical Manager for Test (TMT) whose responsibility it is to define the test scope. The TMT acts thereby as a link between RE and ST. However, testers need to pull information from TMT, rendering the spread of knowledge on requirements changes and matter of individual initiative, that is, person dependent.

Panel B in Figure 4.6 shows that Feature Entity Descriptions, describing the system functionality on a compound level, are used by the SVT to interpret the Test Results and Execution. However, Feature Entity Descriptions are written late in the project, by external consultants, describing how the system is actually implemented. RE states that this is a local sub-optimization that saves effort in the RE department, affects however the work of ST. To render Feature Entity Descriptions useful to ST, they should be created and maintained during the project as the implementation stabilizes. To summarize, the following improvement opportunities were identified:

- Test Cases are created by using different sources of information which are potentially inconsistent.
- Test coverage is asserted manually by a dedicated role since tool support is not flexible enough to maintain traceability over time.
- Feature Entity Descriptions should be written earlier such that they are of more use to ST.

### 5.1.2 Impact on REST-bench

We intended to collect and analyze both artifacts (tangible information) and events when information is exchanged informally, e.g. in ad-hoc meetings, email and on-line conversations, or phone calls. However, eliciting each individual instance of informal communication seemed unrealistic for the targeted effort budget (6–12 person-hours) for data elicitation. The positive results of this first assessment, i.e. the identification of issues, supported our intuition that we already collect enough data that can be efficiently analyzed in the assessment workshop and lead to concrete improvement suggestions.

Regarding the relationships between artifacts, we aimed at producing rich analysis points, identifying *what* information is used by *whom* to create *which* artifact. Therefore we focused to elicit information regarding “used-to-create” relationships. However, using an artifact to create another does not necessarily mean that the two artifacts are linked, rendering the REST

taxonomy heuristics Unterkalmsteiner et al. [2014a] more difficult to apply. Therefore we decided to focus on “linked-to” relationships in the data elicitation of the next cases.

Finally, we relied exclusively on note taking during the interviews. Even though this sped up the artifact map creation process, much time during the assessment workshop was spent on correcting the map. Hence we decided to audio record future interviews and workshops, given that participants gave consent.

## 5.2 Company B

In Company B we could not identify a single project, since Scrum teams (one business analysis and two development/test teams) work continuously on a product, releasing a new version every two months. We chose therefore the last release-cycle as time-frame for the assessment and selected a requirements engineer and a test lead as interviewees. The total staff consisted of 20 engineers while the assessment took place in March 2013.

### 5.2.1 Assessment Results

Regulatory requirements require that the development process in Company B provides traceability that illustrates that safety risks are considered in the requirements, and that the developed services or products comply to these requirements (verification). However, maintaining traceability over time is complicated by storing information in different systems (word documents, a to-do management tool, Team Foundation Server (TFS)). This “infrastructure gap” requires that certain traceability links are maintained manually by referring to record identifiers, leading to increased effort and vulnerability to inconsistencies. This issue is reinforced if traceability links are redundant. For example, looking at Panel A in Figure 4.7, both Product Backlog Item and Use Case refer to a particular risk, maintained manually in both documents. Inconsistencies may be caught (early) during the review of the use case or (late) when the sprint is approved. This redundancy is however no problem (it may indeed ease the location of information), if the links are maintained by the system in which the information is stored. This is the case, for example, for the test artifacts (Test Template, Test Item, Test Protocol) linked to the Product Backlog Item, which are all stored within TFS.

While traceability is generally established with unique references, Customer Validation Tests are linked implicitly to Use Cases by naming conventions (Panel B in Figure 4.7). The Test Lead (TL) maintains the link, but staff turnover may lead to inconsistencies in the future. Furthermore, as soon as the complexity of the validation tests rises, a more sophisticated

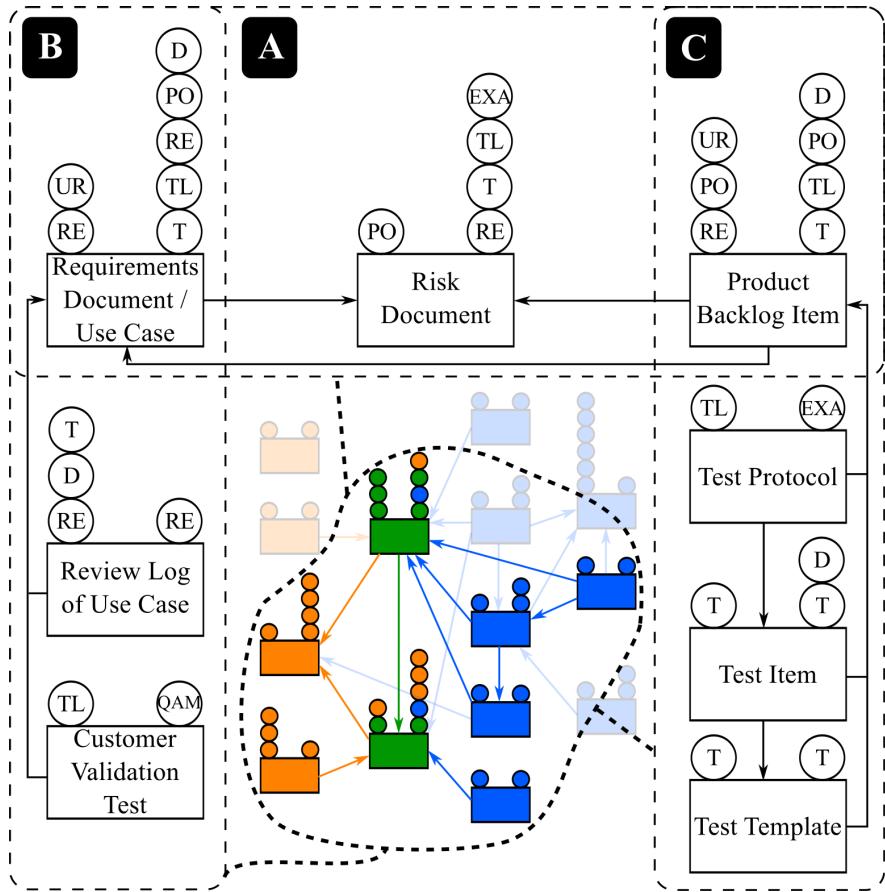


Figure 4.7: Artifact map of Case B

tracing mechanism may be required in order to keep track whether and how the Use Case is covered by the executed scenarios in the Customer Validation Tests.

Looking at Panel B in Figure 4.7, we can observe that ST roles are involved in reviewing documents created by RE. There is however no such involvement of RE roles in reviewing documents created by ST (Panel C). RE does occasionally, when a complex function needs to be verified, provide feedback to ST on particular test cases. Such ad-hoc reviews do however not verify the test scope. This could be achieved by having a review of the Test Template and the test-cases therein (early, before the tests are actually executed), or of the Test Protocol (late, when the test results are available). To summarize, the following improvement opportunities were identified:

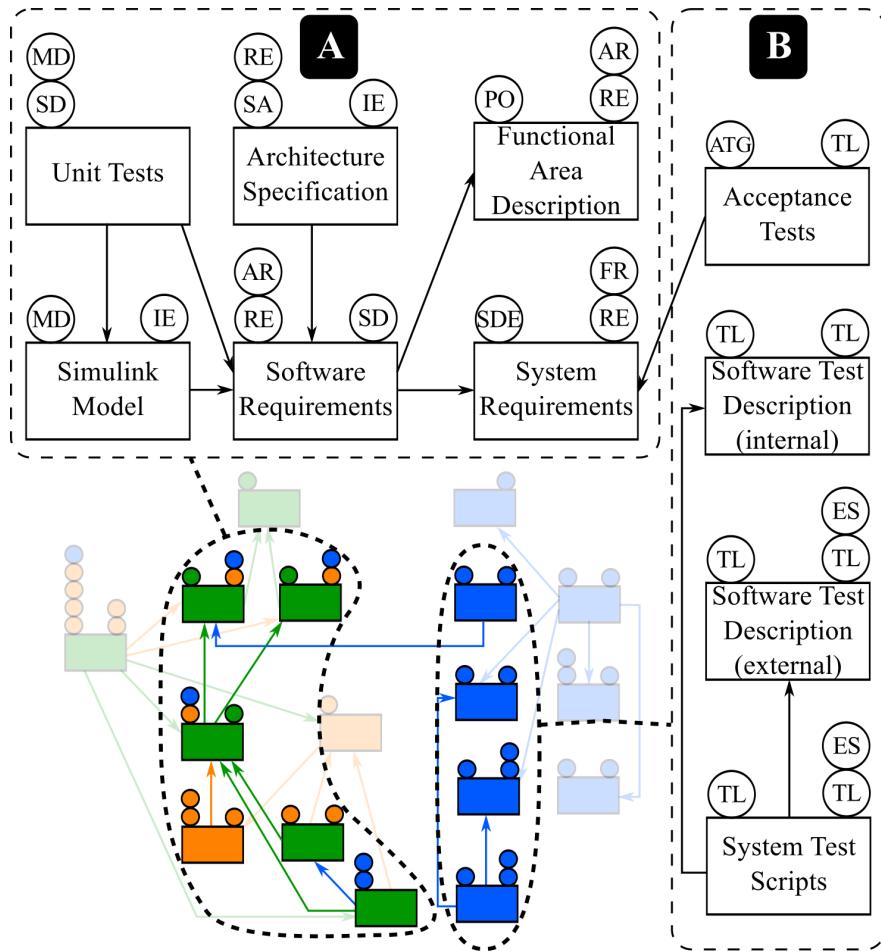


Figure 4.8: Artifact map of Case C

- Manual maintenance of redundant traceability links may lead to inconsistencies.
- Customer Validation Tests are not explicitly mapped to Use Cases.
- RE is little involved in reviewing document created by ST.

### 5.3 Company C

We selected a requirements engineer/software architect with 3 years experience in his current role as RE representative. For the ST representative we chose a test lead with 1.5 years experience. The assessment took place in March 2013, approximately at halftime of the expected three year project duration. The selected project had a staffing of 9 engineers, operating in

a Scrum team. The overall development process in Company C is plan-driven, posing challenges to the project that iterates in two week sprints. The project was responsible for approximately 300 system requirements.

### 5.3.1 Assessment Results

An aspect that is not visible in the artifact map is the project life-cycle and how it influences the artifacts that are created and used. This project is estimated to last three years in total. This investigation took place approximately 1.5 years into the project, at a time when the high-level requirements (System requirements and Functional Area Descriptions, see Panel A in Figure 4.8) were stabilized and soon to be frozen. Due to the volatility of the requirements in the initiation phase of the project, the development team focused on establishing the feasibility of the product, adapting continuously on changing requirements. This led to postponing the formalization of documentation and links, since this process is time consuming and there is a resistance to start it before there is an indication that the system is actually working. Therefore, even if Panel A in Figure 4.8 suggests that traceability from high-level requirements down to unit-tests is given, this is only a goal that has not yet been achieved at this stage of the project. In particular, the links from the Software Requirements to System Requirements and Functional Area Descriptions are in the process of being re-engineered and implemented in the requirements specification tool (SpecTool).

Looking at Panel B in Figure 4.8, it is apparent that there is a gap between the integration tests (Software Test Description artifact) and the requirements they are intended to verify (System requirements and Functional Area Description). Software Requirements have, according to RE, a 1-to-1 mapping to unit-tests. On the other hand, System Test Descriptions, which specify how and what is tested in integration, are currently not linked to any requirements documentation. Both RE and ST agree that linking the system test descriptions to system requirements and functional area descriptions would provide benefits:

- ability to perform requirements coverage analysis
- change impact analysis, leading to reduced test-time
- allow reviews in which both RE and ST participate and validate the effectiveness of integration tests

In this project, the software team started to integrate System Requirements and Functional Area Descriptions in SpecTool, allowing them to link their information to Software Requirements. Although this is an improvement (as manual linking to word documents is not necessary anymore), linking the requirements to the integration tests is still a challenge, since test artifacts are stored in a separated infrastructure, requiring a manual

linking and maintenance process. According to RE, the link implementation in SpecTool by itself may turn out to be problematic. Even minor changes in parent documents, such as correction of spelling mistakes or improving descriptions, trigger a change event, leading to child documents to be flagged as outdated. There is no possibility in SpecTool to qualify the level of a change. That may be a risk, leading to a lower quality of documentation. To summarize, the following improvement opportunities were identified:

- Time consuming re-engineering of Software Requirements and linking to other artifacts.
- Lack of traceability between integration tests and requirements they are intended to verify.

### 5.3.2 *Impact on REST-bench*

As we assessed Company B and C in parallel, we summarize here the lessons learned from both REST-bench applications. As a major change, we reintroduced the elicitation of “used-to-create” relationships between artifacts, adding how information from one artifact is used to create another artifact. This differs from the “linked-to” semantics and is useful as it provides a dynamic aspect to artifact relationships that is lost when considering only links between artifacts. For example, consistency between artifacts is much more difficult to achieve and maintain when there exist “used-to-create” but no “linked-to” relationships between artifacts.

Nevertheless, the assessment in Company C illustrated also one of the weaknesses of a static artifact map. At different points in time, the particular use of an artifact may change in a project. This dynamism is not visible in an artifact map, but can be captured during data collection and then considered while discussing the map at the assessment workshop.

## 5.4 *Company D*

We selected a requirements engineer and a system designer, with 10 and 5 years experience respectively, as RE representatives. We chose a test lead with 5 years experience as representative for the ST perspective. The selected project had a four year duration, involving in peak times about 1000 hard-and software engineers. During the assessment on April 2013, the project was in the closing phase. The project handled approximately 2000 system requirements and 500 functional test cases.

### 5.4.1 *Assessment Results*

Implementation Proposals [Fricker et al., 2010] are typically written as “change this existing functionality like that”, and are used by system de-

signers to architect the software and by ST to develop the test plan (see Panel A in Figure 4.9). Function Descriptions, on the other hand, are written when the software has been implemented, providing a complete description of a function, and are used by ST to write test specifications. ST reported that the information in the Implementation Proposals and the respective Function Descriptions may be inconsistent, leading to situations where they attempted to test functionality that is not supported by the delivered software. Inconsistencies stem from the fact that Function Descriptions are created late in the project forcing ST to use the Implementation Proposal to write test-specifications, and that no role has been assigned the responsibility to check and maintain consistency between Implementation Proposals and Function Descriptions.

Due to the amount of Implementation Proposals and Functional Descriptions per project, maintaining consistency among them is a challenge and may not even be possible with reasonable effort. One possible improvement would be to implement a mechanism that alerts the reader of potential inconsistencies. Furthermore, the responsibility (who, why) and the procedure (what, when) of updating/creating Function Descriptions should be defined. The Implementation Proposal is written by both RE and ST, whereby ST is responsible for the test specific parts. However, the distance in time in which different parts in the Implementation Proposals are written may be very long (up to one year): while RE is working on an Implementation Proposal, ST might be heavily involved in testing, leading to the situation that ST is actually not collaborating with RE in the specification. Then, when ST is ready to specify the testing part, the document may be out-of-date as it has not been updated by RE during the project. An improvement, related to the process, would be to ensure the commitment of both RE and ST to collaborate at the same time on the Implementation Proposal specification.

ST explained that in their work process, they map the requirements (from the Software Design Requirements Specification) to the test cases they execute. In the Test Report they document test results and list which requirements are covered by the executed test cases (see Panel A in Figure 4.9). There is however no feedback loop from ST to RE, leading to the inability to improve low quality requirements when they are marked as un-testable in the Test Report. Furthermore, test-coverage can not be efficiently measured. A requirement to test-case mapping would have the following benefits:

- Regression tests could be efficiently defined (assuming that Software Design Requirements Specification is traced to requirements at higher abstraction levels).

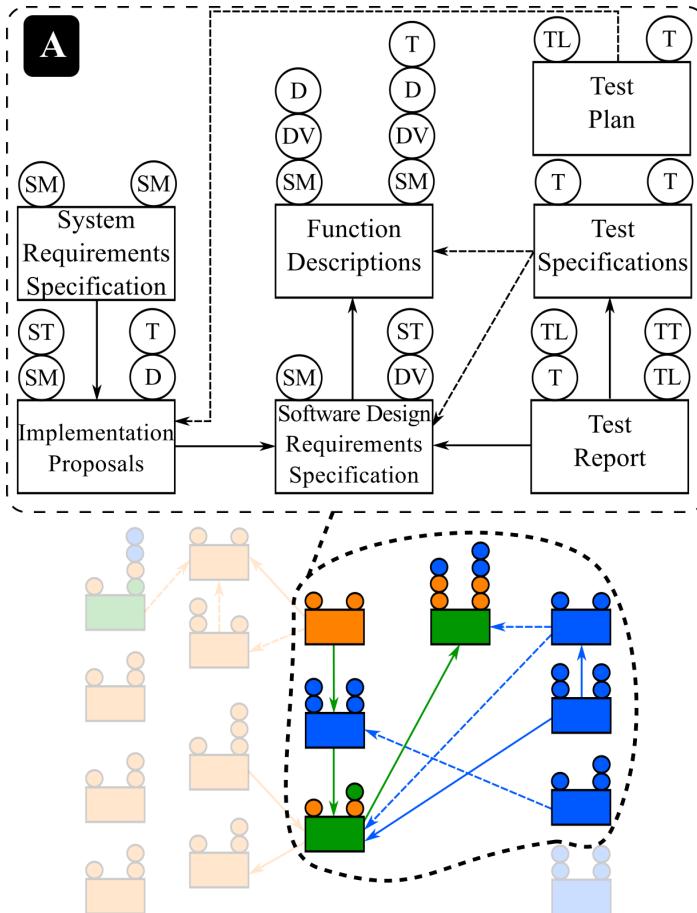


Figure 4.9: Artifact map of Case D

- Change requests involving the addition or removal of requirements would be visible on the ST side, i.e. it would be transparent whether new test cases are needed, or old ones can be removed.
- Statements of verification could be generated without manual effort.

Looking at the artifact map in Figure 4.9, one can observe that ST is, except in the definition of Implementation Proposals where the benefit is immediate, not involved in RE activities. Previous efforts to include ST failed, mostly due to scheduling issues. With the initiative to improve the quality of the Software Design Requirements Specification, new opportunities for ST involvement arise. For example, ST could provide useful input on the requirements quality by looking at their Test Reports. Not testable requirements identified in these reports can either be improved or even removed from the database. A second opportunity for ST involvement

arises as soon as the requirements to test case mapping is implemented. New requirements need to be reviewed and mapped to either existing test cases or the decision needs to be taken to create new test cases. ST could be assigned this responsibility. Then, with the requirements to test case mapping in place, ST can provide feedback, during and after the testing phase, to RE on the quality of requirements, preventing the decay of requirements quality. To summarize, the following improvement opportunities were identified:

- Maintaining consistency between Improvement Proposals and Function Descriptions in order to support ST in testing correct functionality.
- Schedule collaboration between RE and ST such that work on common artifacts leads to coordination rather than misunderstandings.
- Map test cases to requirements such that faults in requirements identified by ST can be fed back to RE.

#### 5.4.2 Impact on REST-bench

In Company D, we analyzed the lack of mapping between individual requirements and test cases. Even though requirements are linked in the Test Report to individual test cases, RE could not benefit from that information as no bi-directional links existed. This observation, together with the analysis of Company C (see Section 5.3.1), where we did not differentiate between uni-directional and bi-directional relationships, led us to the introduction of the “mapped-to” relationship. The semantics of the “mapped-to” relationship allows us to describe links between artifacts that can be followed from either side (corresponding to backward and forward traceability).

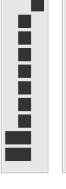
## 6 DISCUSSION

In this section we answer the research questions stated in Section 3.

### 6.1 RQ1: To what extent are the dyad structures from the REST taxonomy useful to elicit improvement opportunities?

Table 4.4 illustrates the mapping between the seeding questions we used to prepare the assessment workshops and the dyad structure properties, summarized in Table 4.1, we identified in our earlier work [Unterkalmsteiner et al., 2014a] (see Chapter 3). We established the mapping in Table 4.4 in order to understand which dyad structure properties are actually useful in generating analytical questions regarding the elicited artifact maps. While properties  $P_1$ ,  $P_2$ ,  $P_5$  and  $P_6$  were useful, we could not yet derive ques-

Table 4.5: Post assessment questionnaire

| Open ended questions  |   |
|---|---|
| Q1  | Did the elicitation (interview session), the artifact map or the collaborative workshop reveal something new, you did not know before, w.r.t. activities, responsibilities or artifacts in: requirements engineering, software testing, the interplay between both?   |
| Q2  | Given the resources you have invested, would you consider to use this method as a complement to project post-mortems to improve the coordination between RE and ST? Please motivate, why yes/no/maybe.  |
| Q3  | Did you discuss the coordination between RE and ST in your organization already before this assessment? If yes, which specific issue(s) have been discussed and why?  |
| Q4  | In general, to elicit more relevant and precise information, would you suggest to include another role (e.g. Software Architect, Developer) in the assessment? Please motivate, why yes/no/maybe.   |
| 5 point Likert scale with possibility to provide explanatory feedback |   |
| Q5  | The assessment (including elicitation and workshop) is an <i>efficient</i> <sup>1</sup> mean to identify issues in relation to the coordination between Requirement Engineering and Software Test.<br>                       |
| Q6  | The assessment (including elicitation and workshop) is an <i>effective</i> <sup>2</sup> mean to identify issues in relation to the coordination between Requirement Engineering and Software Test.<br>                       |
| Q7  | The artifact map and the discussion in the workshop increased my understanding of the coordination between RE and ST in the studied project.<br>   |
| Q8  | The artifact map and the discussion in the workshop increased my awareness for potential waste (e.g. unused documentation).<br>  |
| Q9  | The artifact map and the discussion in the workshop increased my awareness for potential gaps in the coordination between RE and ST.<br>   |
| Q10   | Interactions (scheduled meetings, but also casual encounters over coffee, in the corridor or at the office door) are equal or even more important as written documentation for the successful execution of the project.<br> |

<sup>1</sup> Efficiency in general describes the extent to which time or effort is well used for the intended task or purpose.

<sup>2</sup> Effectiveness is the capability of producing a desired result. When something is deemed effective, it means it has an intended or expected outcome.  
<sup>3</sup> The bars represent the 10 responses from the study participants on a Likert scale (strongly agree, agree, neutral, disagree, strongly disagree)

tions by using properties  $P_3$  and  $P_4$  (see Table 4.4 for a description of these properties). However, this does not preclude the possibility that in future assessments those properties might generate useful seeding questions.

Looking at the overall focus of REST-bench, it seems straightforward why no questions regarding  $P_3$ , intermediate nodes, were generated: the data elicitation is geared towards artifacts that are used and created by requirements engineers and software testers (see Section 4.2). Q4 in our post assessment questionnaire addressed this specific aspect, i.e. whether other roles (e.g. software architects, developers) should be included in the assessment. Nine out of ten participants suggested including other roles (six argued for software developers, one for a configuration manager, two for no specific role). While we decided not to include more roles, in order to keep the overall assessment effort low, one could argue that a developer participating in the assessment workshop alone would be beneficial as a user of requirements and creator of the system under test, thereby contributing to the identification of improvement potential.

The second dyad structure property for which we could not generate analytical questions is  $P_4$  – RE and ST node proportion. This property expresses the relative effort spent in creating and maintained RE and ST artifacts respectively. However, we did not find any evidence that this proportion could indicate an issue in REST alignment or serve as an analytical lever to identify improvement opportunities.

To gauge the overall relevance and usefulness of REST-bench, we asked questions Q1–Q3 (see Table 4.5) to participants. Regarding the relevance of REST alignment (Q3), eight out of ten participants reported that they had in the past discussions on how to better coordinate requirements engineering and testing activities. As one participant put it, “*we have had some discussions in my team how to get the correct information when we need it from RE. But we have not discussed that with the RE organization yet.*” This indicates that REST-bench addresses an industry need, providing structure and guidelines and thereby lowering the barrier to perform such an assessment. All ten participants stated that they would use REST-bench as a complement to project postmortems (Q2), as “*it gives a complete picture of the artifacts, the relations between them and a good material for use in discussions about potential process improvements*”, and “*is a very good and easy way to ensure that all involved use and acknowledge each others documents and their purpose.*” This is further corroborated by the results on Q5 and Q6, where we asked the participants whether REST-bench is efficient and effective (see Table 4.5). One participant stated that “*the method gives more truth on what is actually used. We already have processes for a lot of things but these are somewhat theoretical and not adapted per project.*” On the other hand, another participant stated that “*it is not really guaranteed that we take action just because we have identified the issues.*” With respect to Q1, whether participants learned something new from the artifact map or the assessment workshop, eight

out of ten participants indicated that they identified aspects in the way of working they were not aware of. This is further supported by the results to Q7 in Table 4.5, even though a participant stated that “*I believe I already had a pretty good overview of the situation. Probably because we are a relatively small team where everybody works with a broad range of tasks*”, and some other stated that “*it confirmed my thoughts.*” Another aspect we investigated is whether REST-bench increases the awareness of potential waste (Q8) or gaps (Q9) in the coordination between RE and ST. The results (see Table 4.5) suggest that the potential is more geared towards identifying gaps than waste. REST-bench can be used to identify waste candidates [Khurum et al., 2014]. However, to actually remove waste all project stakeholders are necessary. This corroborates the result from Q4 discussed earlier, including developers could improve the potential of REST-bench to eliminate waste.

## 6.2 RQ2: To what extent is REST-bench in Agile and plan-driven environments useful?

One of our concerns when we planned the validation of REST-bench was the methods’ focus on using documentation as a proxy to determine REST alignment. Indeed, in Case A described in Section 5.1, we planned to complement the data collection with recording instances of informal communication, however rejected the idea due to the involved effort and inaccuracy to do that manually. Therefore, we questioned whether we can apply REST-bench in Agile environments at all, and collect relevant data and produce useful results. Agile approaches are notorious for promoting as little documentation as possible [Fowler and Highsmith, 2001, Chau et al., 2003, Cohen et al., 2004, Nerur et al., 2005].

In order to understand whether there is a difference in usefulness of REST-bench, we analyzed the post assessment questionnaire by stratifying the answers into plan-driven and Agile groups, using the project characteristics illustrated in Table 4.2. There were five participants in each group. The largest divergence can be observed in Q8 (REST-bench increases the awareness for potential waste). The Agile group tends to disagree (██████), while the plan-driven group tends to agree (██████). On the other hand, both the Agile (██████) as well the plan-driven (██████) group agree that REST-bench increases the awareness of gaps in the coordination between RE and ST (Q9). This conforms to the observations by Stettina and Heijstek [2011] where practitioners report that Agile projects are often too little documented. Both groups show similar tendencies to the remaining questions:

- Q5 – efficiency of REST-bench: Agile (██████) and plan-driven (██████)

- Q6 – effectiveness of REST-bench: Agile ( ) and plan-driven ( )
- Q7 – increased understanding: Agile ( ) and plan-driven ( )
- Q10 – importance of informal interactions: Agile ( ) and plan-driven ( )

These results suggest that REST-bench is useful both in Agile and plan-driven environments. This is further corroborated by the actual case assessment results, where for both Agile and plan-driven projects relevant improvement opportunities were identified.

### 6.3 RQ3: Is REST-bench usable?

We answer this question from the perspective of the researcher who applied the method. REST-bench has not been transferred to an industry partner so that ease of use from the perspective of a practitioner can not be validated yet. However, the results from the questionnaire indicate that REST-bench is, from a participants' perspective, efficient and effective (see Q5 and Q6 in Table 4.5).

We look at three aspects of usability, breaking it down to each step in REST-bench: required up-front investment, the effort to perform, and the support REST-bench provides in each step. With up-front investment we mean the one-time cost for the analyst to learn a particular step. In Table 4.6, we provide the relative cost of each step. In total, we estimate the up-front cost to learn the REST-bench method to 8–12 hours.

The required estimated effort is shown in the third column in Table 4.6, and further motivated in the respective sections where each step is explained (Section 4.1 – 4.5). We regard these estimates as upper limits (with the suggested number of participants), if the assessment is performed in

Table 4.6: Usability assessment of REST-bench

| Step                                  | Cost | Effort (p-h) | Support within REST-bench                  |
|---------------------------------------|------|--------------|--|
| 1. Selection                          | 5%   | 2–4          | heuristics on participant profile          |
| 2. Data elicitation                   | 15%  | 6–12         | defined procedure / templates              |
| 3. Map construction                   | 40%  | 8–12         | seeding questions / mapping tool prototype |
| 4. Collaborative issue identification | 30%  | 6–10         | defined procedure                          |
| 5. Report recommendations             | 10%  | 8–12         | report structure                           |

regular intervals. A common detractor for conducting postmortems is lack of time [Keegan and Turner, 2001, Glass, 2002a]. Therefore, we designed REST-bench to be efficient, while still providing value to the organization conducting the assessment. This efficiency is achieved by providing guidelines and heuristics within REST-bench (see fourth column in Table 4.6). Furthermore, each of the steps is supported by a series of examples presented in this chapter.

## 7 CONCLUSIONS

The chapter presents a method to identify improvement opportunities in the coordination between requirements engineering (RE) and software testing (ST). The method, REST-bench, is based on the premise that information, the way it is created, used and linked, is key for understanding how requirements and test engineers coordinate and how their collaboration can be improved. We used the information dyad and the emerged dyad structures from our earlier work (see Chapter 3) to drive an assessment process that is designed to be lightweight in terms of resource use (30–50 person-hours per assessment).

We validated REST-bench by applying it in five companies. In all assessments we could identify issues that were taken up by the involved companies to trigger improvements. With respect to RQ1, whether the dyad structure properties from the REST taxonomy were useful, we found that all but two properties generated seeding questions that can be used to identify issues and elicit improvement opportunities. Furthermore, the feedback gathered from the assessment participants supports our conclusion that REST-bench is an effective method to identify gaps, and to a lesser degree waste, in the artifacts that support RE and ST coordination. With respect to RQ2, applying REST-bench in Agile and plan-driven contexts, we conclude that the proposed method is useful in both while it is more likely to identify waste in plan-driven contexts. Furthermore, projects with very small teams requiring little coordination will likely not benefit from a REST-bench assessment. Answering RQ3, we found that REST-bench is usable from the perspective of the analyst who conducted the assessments.

To corroborate these results we plan to further validate the usefulness and usability of REST-bench by training practitioners in the autonomous use of the method, supported by efficient tools for data collection and artifact mapping.



# 5

5

---

## AN INDUSTRIAL CASE STUDY ON USING TEST CASES AS REQUIREMENTS

---

**SUMMARY** It is a conundrum that agile projects can succeed “without requirements” when weak requirements engineering is a known cause for project failures. While Agile development projects often manage well without extensive requirements documentation, test cases are commonly used as requirements. We have investigated this agile practice at three companies in order to understand how test cases can fill the role of requirements. We performed a case study based on twelve interviews performed in a previous study. The findings include a range of benefits and challenges in using test cases for eliciting, validating, verifying, tracing and managing requirements. In addition, we identified three scenarios for applying the practice, namely as a mature practice, as a de facto practice and as part of an agile transition. The findings provide insights into how the role of requirements may be met in agile development including challenges to consider.

### 1 INTRODUCTION

Agile development methods strive to be responsive to changing business requirements by integrating requirements, design, implementation and testing processes [Sommerville, 2005, Layman et al., 2006]. Face-to-face communication is prioritized over written requirements documentation and customers are expected to convey their needs directly to the developers [Beck et al., 2001, Ramesh et al., 2010]. However, weak customer communication in combination with minimal documentation is reported to cause problems in scaling and evolving software for agile projects [Ramesh et al., 2010].

Requirements specifications fill many roles. They are used to communicate among stakeholders within a software development project, to drive design and testing, and to serve as a reference for project managers and in the evolution of the system [Davis, 2005]. Due to the central role of requirements in coordinating software development, there exists a plethora

of research on how to document requirements with varying degrees of formality depending on its intended use. This spans from formal requirements specifications [van Lamsweerde, 2000] and requirements models [Pohl, 2010], over templates [Mavin and Wilkinson, 2010] to user stories [Cohn, 2004] and requirements expressed using natural language. At the formal end of the spectrum, requirements specifications can be automatically checked for consistency [Heitmeyer et al., 1996] and used to derive other artifacts, e.g. software designs [Dromey, 2003] or test cases [Miller and Strooper, 2010]. For the less formal approaches, requirements documentation is driven by heuristics and best practices for achieving high quality [Davis et al., 1993] requirements.

The coordination of evolving requirements poses a challenge in aligning these with later development activities including testing [Bjarnason et al., 2014a]. In a previous study we identified the use of test cases as requirements (TCR) as one of several industrial practices used to address this challenge (see Chapter 2). In this chapter, we investigate this practice further by a more detailed analysis of the interview data from the three case companies (of six) that explicitly mentioned this practice. The case study presented in this chapter investigates how the practice may support the role of requirements engineering (RE) by investigating *RQ1* How does the TCR practice fulfill the role of requirements? and *RQ2* Why and how is the TCR practice applied?

The rest of this chapter is organized as follows. Section 2 describes related work. Section 3 presents the case companies and Section 4 the applied research method. The results are reported in Section 5, while the research questions are answered in Sections 6 and 7. The chapter is concluded in Section 8.

## 2 AGILE RE: TEST CASES AS REQUIREMENTS DOCUMENTATION

In agile software development requirements and tests can be seen as two sides of the same coin. Martin and Melnik [2008] hypothesize that as the formality of specifications increases, requirements and tests become indistinguishable. This principle is taken to the extreme by unit tests [Whittaker, 2000] where requirements are formalized in executable code. Practitioners report using unit tests as a technical specification that evolves with the implementation [Runeson, 2006]. However, unit tests may be too technical for customers and thereby lack the important attribute of being understandable to all relevant stakeholders.

Acceptance tests are used to show customers that the system fulfills the requirements [Hsia et al., 1997]. However, developing acceptance tests from requirements specifications is a subjective process that does not guarantee that all requirements are covered [Hsia et al., 1997]. This is further compli-

cated by requirements documentation rarely being updated [Lethbridge et al., 2003], leading to potentially outdated acceptance tests. In agile development, automated acceptance tests (AATs) drive the implementation and address these issues by documenting requirements and expected outcomes in an executable format [Ramesh et al., 2010, Haugset and Hanssen, 2008]. This agile practice is known, among others, as customer tests, scenario tests, executable/automated acceptance tests, behavior driven development and story test driven development [Park and Maurer, 2010].

Some organizations view and use the AATs as requirements thereby fully integrating these two artifacts [Martin and Melnik, 2008]. AATs are used to determine if the system is acceptable from a customer perspective and used as the basis for customer discussions, thus reducing the risk of building the wrong system. However, the communication might be more technical and require more technical insight of the customer. Melnik et al. [2006] found that customers in partnership with software engineers could communicate and validate business requirements through AATs, although there is an initial learning curve.

The conceptual difficulty of specifying tests before implementation [Causovic et al., 2011, George and Williams, 2004, Janzen and Saiedian, 2007] led to the conception of behavior-driven development (BDD) [North, 2006]. BDD incorporates aspects of requirements analysis, requirements documentation and communication, and automated acceptance testing. The behavior of a system is defined in a domain-specific language (DSL); a common language that reduces ambiguities and misunderstandings. This is further enhanced by including terms from the business domain in the DSL.

Haugset and Hanssen studied acceptance test driven development (ATDD) as an RE practice and report on its benefits and risks [Haugset and Hanssen, 2008]. Our work extends on this by also investigating companies that use the TCR practice without applying ATDD principles.

### 3 CASE COMPANIES

The three case companies all develop software using an agile development model. However, a number of other factors vary between the companies. These factors are summarized in Table 2.1 in Chapter 2 (Company A, B and F) and the interviewees are characterized in Table 5.1.

**COMPANY A** develops network equipment consisting of hardware and software. The software development unit covered by the interview study has around 150 employees. The company is relatively young but has been growing fast during the past few years. A typical software project has a lead time of 6–18 months, around 10 co-located members and approx-

Table 5.1: Interviewees per company. Experience in role noted as S(enior) = more than 3 years, or J(unior) = up to 3 years. Interviewees mentioning the TCR practice are *emphasized*. Note: For Company B, software developers also perform RE and testing tasks.

| Role                  | Company A        | Company B | Company F                           |
|-----------------------|------------------|-----------|-------------------------------------|
| Requirements engineer |                  |           | F1:S, F6:S, F7:S                    |
| Systems architect     |                  |           | F4:S                                |
| Software developer    | B1:J, B2:S, B3:S | F13:S     |                                     |
| Test engineer         | A2:S             |           | F9:S, F10:S, F11:J,<br>F12:S, F14:S |
| Project manager       | A2:S             |           | F3:J, F8:S                          |
| Product manager       | A4:S             |           |                                     |
| Process manager       |                  |           | F2:J, F5:S, F15:J                   |

imately 100 requirements and 1,000 system test cases. A market-driven requirements engineering process is applied. The quality focus for the software is on availability, performance and security. Furthermore, the company applies a product-line approach and uses open-source software in their development. A product manager, a project manager, and a tester were interviewed at Company A, all of which described how the company manages requirements as test cases.

COMPANY B is a consultancy firm that provides technical services to projects that vary in size and duration. Most projects consist of one development team of 4–10 people located at the customer site. The requirements are defined by a customer (bespoke). The three consultants that were interviewed at Company B can mainly be characterized as software developers. However, they all typically take on a multitude of roles within a project and are involved throughout the entire life-cycle. All three of these interviewees described the use of the TCR practice.

COMPANY F develops software for embedded products in the telecommunications domain. The software development unit investigated in this study, consists of 1,000 people. At the time of the interviews, the company was transitioning from a waterfall process to an agile process. Projects typically run over 2 years and include 400–500 people. The project size and lead time is expected to decrease with the agile process. The projects handle a combination of bespoke and market-driven requirements. Including the product-line requirements, they handle a very complex and large set of requirements. Six of the interviewees (of 15) discussed the practice, namely

one requirements engineer, two project managers, two process managers and one tester.

#### 4 METHOD

We used a flexible exploratory case study design and process [Runeson et al., 2012] consisting of four stages: 1) Definition, 2) Evidence selection, 3) Data analysis and 4) Reporting.

5

**DEFINITION OF RESEARCH QUESTIONS AND PLANNING.** Since we were interested in how agile development can be successful “without requirements” we selected to focus on the practice of using test cases as requirements. We formulated the research questions, (RQ1) How does the TCR practice fulfill the role of requirements? and (RQ2) Why and how is the TCR practice applied?

**EVIDENCE SELECTION.** We selected to use word-by-word transcriptions from our previous study of RE-Testing coordination. The research questions of this chapter are within the broader scope of the previous study (see Chapter 2), which also included agile processes. In addition, the semi-structured interviews provided rich material since the interviewees could freely describe how practices were applied including benefits and challenges. Data selection was facilitated by the rigorous coding performed in the previous study. We selected the interview parts coded for the TCR practice. In addition, the transcripts were searched for key terms such as “acceptance test”, “specification”.

**DATA ANALYSIS.** The analysis of the selected interview data was performed in two steps. First the transcripts were descriptively coded. These codes were then categorized into benefits and challenges, and reported per case company in Section 5. The analysis was performed by the first author. The results were validated independently by the third author. The third author analyzed and interpreted a fine-grained grouping of the interview data produced in the previous study, and compared this against the results obtained by the first researcher. No conflicting differences were found.

#### 5 RESULTS

Two of the investigated companies apply the TCR practice while the third company plan to apply it. The maturity of the practice thus varied. The interviewees for Company B provided the most in depth description of the practice, which is reflected in the amount of results per company. Limitations of the findings are discussed at the end of this section.

### *5.1 Company A: A De Facto Practice*

Test cases have become the de facto requirements in company A due to weak RE (A2<sup>1</sup>), i.e. the RE maturity in the company is low while there is a strong competence within testing. Formal (traditional) requirements are mainly used at the start of a project. However, these requirements are not updated during the project and lack traceability to the test cases. Instead, the test cases become the requirements in the sense that they verify and ensure that the product fulfills the required behavior.

**BENEFITS.** Efficient way of managing requirements in a small and co-located organization that does not require managing and maintaining a formal requirements specification once test design has been initiated (A1). In addition, the structure of the test specifications is closer to the code simplifying navigation of these “requirements” once the implementation has started (A1).

**CHALLENGES.** As the company grows, the lack of traces to formal requirements is a problem in communication of requirements changes to the technical roles (A1, A2) and in ensuring correct test cases (A2). In addition, the test cases lack information about requirements priority, stakeholders etc., needed by the development engineers when a test case fails (A2) or is updated (A3). The untraced artifacts do not support either ensuring test coverage of the formal requirements (A1, A3), or identifying the test cases corresponding to the requirements re-used for a new project (A2).

### *5.2 Company B: An Established Practice*

Company B actively applies the TCR practice through behavior-driven development supported by tools. The customer and the product owner define product and customer requirements. Then, for each iteration, the development engineers produce acceptance criteria (user scenarios) and acceptance test cases from these requirements. These “requirements test cases” are iterated with the business roles to ensure validity (B1), and entered into an acceptance test tool that produces an executable specification. The interviewees described that the acceptance criteria can be used as a system specification. However, interviewee B<sub>3</sub> stated that the acceptance criteria can be read “to get an impression. But, if you wonder what it means, you can look at the implementation”, i.e. this documentation is not fully stand-alone.

---

<sup>1</sup> Mentioned by this interviewee, see interviewee codes in Table 5.1

**BENEFITS.** The interviewees stated that the main benefits are improved customer collaboration around requirements, strengthened alignment of business requirements with verification, and support for efficient regression testing. The customer collaboration raises the technical discussion to a more conceptual level while also improving requirements validity, since, as an engineer said, “we understand more of the requirements. They concretize what we will do” (B1). This alignment of business and technical aspects was experienced to also be supported when managing requirements changes by the use of acceptance test cases as formal requirements (B2, B3). At the end of a project the acceptance test cases show “what we’ve done” (B2). Furthermore, the executable specification provided by this practice, in combination with unit tests, acts as a safety net that enables projects to “rebound from anything” (B1) by facilitating tracking of test coverage, efficiently managing bugs and performance issues.

**CHALLENGES.** The interviewees mentioned several challenges for the practice concerning active customer involvement, managing complex requirements, balancing acceptance vs. unit tests and maintaining the “requirements test cases”. Over time the company has achieved active customer involvement in defining and managing requirements with this practice, but it has been challenging to ensure that “we spoke the same language” (B3). The interviewees see that customer competence affects the communication and the outcome. For example, interviewee B3 said that non-technical customers seldom focus on quality requirements. Similarly, getting the customer to work directly with requirements (i. e. the acceptance test cases) in the tool has not been achieved. This is further complicated by issues with setting up common access across networks.

Complex interactions and dependencies between requirements, e. g. for user interfaces (B1) and quality requirements (B2), are a challenge both to capture with acceptance test cases and in involving the customer in detailing them. Furthermore, automatically testing performance and other quality aspects on actual hardware and in a live testing environment is challenging to manage with this approach.

All interviewees mentioned the challenge in balancing acceptance vs. unit test cases. It can be hard to motivate engineers to write acceptance-level test cases. Furthermore, maintenance of the acceptance test cases needs to be considered when applying this practice (B1, B2, B3). Interviewee B3 pointed out that test cases are more rigid than requirements and thus more sensitive to change. There is also a risk of deteriorating test case quality when testers make frequent fixes to get the tests to pass (B2).

### 5.3 *Company F: Planned Practice as part of Agile Transition*

The agile transition at the company included introduction of this practice. Requirements will be defined by a team consisting of a product owner, developers and testers. User stories will be detailed into requirements that specify “how the code should work” (F8). These will be documented as acceptance test cases by the testers and traced to the user stories. Another team will be responsible for maintaining the software including the user stories, test cases and traces between them. In the company’s traditional process, test cases have been used as quality requirements, as a de facto practice. Interviewee F1 describes an attempt to specify these as formal requirements that failed due to not reaching an agreement on responsibility for the cross-functional requirements within the development organization.

**BENEFITS.** The practice is believed to decrease misunderstandings of requirements between business and technical roles, improve on the communication of changes and in keeping the requirements documentation updated (F5, F10).

**CHALLENGES.** Integrating the differing characteristics and competences of the RE and testing activities are seen as a major challenge (F5, F10) in the collaboration between roles and in the tools. RE aspects that need to be provided in the testing tools include noting the source of a requirement, connections and dependencies to other requirements and validity for different products (F5).

### 5.4 *Limitations*

We discuss limitations of our results using guidelines provided by Runeson et al. [2012].

**CONSTRUCT VALIDITY.** A main threat to validity lies in that the analyzed data stems from interviews exploring the broader area of coordinating RE and testing. This limits the depth and extent of the findings to what the interviewees spontaneously shared around the practice in focus in this chapter. In particular, the fact that the practice was not yet fully implemented at Company F at the time of the interviews limits the insights gained from those interviews. However, we believe that the broad approach of the original study in combination with the semi-structured interviews provide valuable insights, even though further studies are needed to fully explore the topic.

**EXTERNAL VALIDITY.** The findings may be generalized to companies with similar characteristics as the case companies (see Section 3 by theoretical generalization [Runeson et al., 2012]).

**RELIABILITY.** The varying set of roles from each case poses a risk of missing important perspectives, e.g. for Company B the product owner's view would complement the available interview data from the development team. There is a risk of researcher bias in the analysis and interpretation of the data. This was partly mitigated by triangulation; two researchers independently performing these steps. Furthermore, a rigorous process was applied in the (original) data collection including researcher triangulation of interviewing, transcription and coding, which increases the reliability of the selected data.

## 6 TEST CASES IN THE ROLE OF REQUIREMENTS (RQ1)

We discuss how the TCR practice supports the main roles of RE and the requirements specification according to roles defined by Lauesen [2002], i.e. the elicitation and validation of stakeholders' requirements; software verification; tracing; and managing requirements. The discussion is summarized in Table 5.2.

### 6.1 *Elicitation and Validation*

The TCR practice supports elicitation and validation of requirements by its direct and frequent communication between business and technical roles for all companies. The customer involvement in combination with increased awareness of customer perspectives among the technical roles supports defining valid requirements. This confirms observations by Melnik and Maurer [2007], Park and Maurer [2009], Haugset and Hanssen [2008] and Latorre [2014]. Furthermore, at Company B, the use of the acceptance criteria format led to customers expressing requirements at a higher abstraction level instead of focusing on technical details. Thus, this practice can address the elicitation barrier of requesting specific solutions rather than expressing needs [Lauesen, 2002].

Nevertheless, the TCR practice requires good customer relations, as stated by interviewees in Company B. Active customer involvement is a known challenge for agile RE due to time and space restrictions for the customer, but also due to that this role requires a combination of business and technical skills [Ramesh et al., 2010, Latorre, 2014]. Business domain tools can be used to facilitate the customers in specifying acceptance tests [Park and Maurer, 2009]. For example, Haugset and Hanssen [2008] report

Table 5.2: Summary of benefits and challenges using TCR per role of RE.

| Benefits                                 | Challenges                             |
|--|--|
| Elicitation and Validation               |  |
| Cross-functional communication           | Good customer-developer relationship   |
| Align goals & perspectives between roles | Active customer involvement            |
| Address barrier of specifying solutions  | Sufficient technical and RE competence |
|  | Complex requirements                   |
| Verification                             |  |
| Supports regression testing              | Quality requirements                   |
| Increased requirements quality           |  |
| Test coverage                            |  |
| Tracing                                  |  |
| Requirements – test case tracing in BDD  | Tool integration                       |
| Requirements Management                  |  |
| Maintaining RET alignment                | Locating impacted requirements         |
| Requirements are kept updated            | Missing requirements context           |
| Communication of changes                 | Test case maintenance                  |
| Efficient documentation updates          |  |

that customers used spread-sheets to communicate information and never interacted directly with actual test cases.

Eliciting and validating requirements, in particular complex ones, relies on competence of the roles involved. At Company B limited technical knowledge affected the customer's ability to discuss quality requirements. This can lead to neglecting to elicit them altogether [Ramesh et al., 2010]. Similarly, capturing complex requirements with acceptance test cases is a challenge, in particular for user interactions and quality requirements.

## 6.2 Verification

The TCR practice supports verification of requirements by automating regression tests as for Company B. The AATs act as a safety net that catches problems and enables frequent release of product-quality code. This was also observed by Kongslí [2006], Haugset and Hanssen [2008] and Latorre [2014]. The practice ensures that all specified requirements (as test cases) are verified and test coverage can be measured by executing the tests.

The verification effort relies on verifiable, clear and unambiguous requirements [Davis, 2005]. Test cases are per definition verifiable and the format used by Company B supports defining clear requirements. Never-

theless, Company B mentioned quality requirements as a particular challenge for embedded devices as this requires actual hardware. This confirms previous findings by Ramesh et al. [2010] and Haugset and Hanssen [2008] that quality requirements are difficult to capture with AATs.

### 6.3 Tracing

Tracing of requirements and test cases is supported by the TCR practice, however the benefits depend on the context. Merely using test cases as de facto requirements (as in Company A) does not affect tracing. For the BDD approach applied at Company B, the tools implicitly trace acceptance criteria and test cases, although there are no traces between the original customer requirements and the acceptance criteria. Hence, as the requirements evolve [Mugridge, 2008] this knowledge is reflected purely in the test cases.

At Company F, where user stories were to be detailed directly into acceptance test cases, tracing remains a manual, albeit straight forward task of connecting acceptance test cases to the corresponding user stories. Furthermore, the responsibility for these traces is clearly defined in the development process, a practice identified by Uusitalo et al. [2008] as supporting traceability. However, it is a challenge for the company to identify tools which provide sufficient support for requirements and for testing aspects, and for the integration of the two.

### 6.4 Requirements Management

The TCR practice provides benefits in managing requirements efficiently throughout the life-cycle. As mentioned for Companies A and B, the practice facilitates a joint understanding of requirements that provides a base for discussing and making decisions regarding changes. However, the practice also requires effort in involving development engineers in the requirements discussion. The optimal balance between involving these technical roles to ensure coordination of requirements versus focusing on pure development activities remains as future work.

The challenge of keeping requirements updated after changes [Bjarnason et al., 2014a] is addressed by a close integration with test cases, as for Company B, since the test cases are by necessity updated throughout the project. Furthermore, since the requirements are documented in an executable format, conflicting new or changed requirements are likely to cause existing test cases to fail. However, locating requirements in a set of test cases was mentioned as a challenge for Company B due to badly structured test cases. The difficulty of organizing and sorting automated tests has also been reported by Park and Maurer [2010].

Contextual requirements information, e.g. purpose and priority [Lauesen, 2002], is seldom retained in the test cases but can support, for example, impact analysis and managing failed test cases. Without access to contextual information from the test cases, additional effort is required to locate it to enable decision making.

## 7 THE REASONS FOR AND CONTEXTS OF THE PRACTICE (RQ2)

Each case company applies the practice differently and for different reasons. At Company A it has become a *de facto practice* due to strong development and test competence, and weak RE processes. However, merely viewing test cases as requirements does not fully compensate for a lack of RE. Company A faces challenges in managing requirements changes and ensuring test coverage of requirements. The requirements documentation does not satisfy the information needs of all stakeholders and staff turnover may result in loss of (undocumented) product knowledge. As size and complexity increase so does the challenge of coordinating customer needs with testing effort [Bjarnason et al., 2014a].

Company B applies the practice *consciously using a full BDD approach including tool support*. This facilitates customer communication through which the engineering roles gain requirements insight. The AATs provide a feedback system confirming the engineers' understanding of the business domain [Park and Maurer, 2009]. However, it is a challenge to get customers to specify requirements in the AAT tools. Letting domain experts or customers provide information via e.g. spread-sheets may facilitate this [Park and Maurer, 2009].

The third practice variant is found at Company F, where it is consciously *planned as part of a transition to agile processes* applying story test driven development [Park and Maurer, 2010]. The practice includes close and continuous collaboration around requirements between business and development roles. However, no specific language for expressing the acceptance criteria or specific tools for managing these are planned. In contrast to the *de facto* context, Company F envisions this practice as enabling analysis and maintenance of requirements. To achieve this, requirements dependencies and priorities need to be supported by the test management tools.

## 8 CONCLUSIONS AND FUTURE WORK

Coordinating and aligning frequently changing business needs is a challenge in software development projects. In agile projects this is mainly addressed through frequent and direct communication between the customer and the development team, and the detailed requirements are often documented as test cases.

Our case study provides insights into how this practice meets the various roles that the requirements play. The results show that the direct and frequent communication of this practice supports eliciting, validating and managing new and changing customer requirements. Furthermore, specifying requirements as acceptance test cases allow the requirements to become a living document that supports verifying and tracing requirements through the life cycle. We have also identified three contexts for this practice; as a de facto practice, part of an agile transition and as a mature practice.

The results can aid practitioners in improving their agile practices and provide a basis for further research. Future work includes investigating how to further improve the RE aspects when documenting requirements as test cases.



# 6

---

## LARGE-SCALE INFORMATION RETRIEVAL IN SOFTWARE ENGINEERING – AN EXPERIENCE REPORT FROM INDUSTRIAL APPLICATION

---

6

**SUMMARY** Software Engineering activities are information intensive. Research proposes Information Retrieval (IR) techniques to support engineers in their daily tasks, such as establishing and maintaining traceability links, fault identification, and software maintenance. We describe an engineering task, test case selection, and illustrate our problem analysis and solution discovery process. The objective of the study is to gain an understanding of to what extent IR techniques (one potential solution) can be applied to test case selection and provide decision support in a large-scale, industrial setting. We analyze, in the context of the studied company, how test case selection is performed and design a series of experiments evaluating the performance of different IR techniques. Each experiment provides lessons learned from implementation, execution, and results, feeding to its successor. The three experiments led to the following observations: 1) there is a lack of research on parameter optimization of IR techniques for software engineering problems; 2) scaling IR techniques to industry data is challenging and for certain techniques still unsolved; 3) the IR context poses constraints on the empirical evaluation of IR techniques, requiring more research on developing valid statistical approaches. We believe that our experiences in conducting a series of IR experiments with industry grade data are valuable for peer researchers so that they can avoid the pitfalls that we have encountered. Furthermore, we identified challenges that need to be addressed in order to bridge the gap between laboratory IR experiments and real applications of IR in the industry.

### 1 INTRODUCTION

The field of Software Engineering thrives on the continuous interchange of knowledge and experience between those who research and those who practice [Wohlin et al., 2012]. Research without application is, in the long-term, meaningless and application without research leads to stagnation

[Ivarsson and Gorschek, 2011]. This view requires that research efforts are seeded by concrete problem statements from industry, and that solutions are evaluated within industrial applications and validated by practitioners [Gorschek et al., 2006]. Scientific advances require also that results are reproducible, which in turn is only possible with disciplined reporting that covers the necessary details to replicate studies. Shepperd et al. [2014] performed a meta-analysis on 600 software defect prediction (SDP) results published in literature and determined that differences in prediction performance varies with the research group rather than the studied SDP technique. They suggest that this researcher bias could be addressed by improving the communication of study designs and details of the used technologies.

On that note, even though we address a different issue than SDP, we report on experiences in identifying, developing and evaluating a potential solution for our industrial partner: test case selection [Rothermel and Harrold, 1996] decision support by Information Retrieval techniques [Grossman and Frieder, 2004]. The main goal of this chapter is to illustrate the path from problem identification to solution definition and evaluation, and to dissect the taken design decisions that address a real-world software engineering problem. With complex problem statements it is sensible to develop new solutions upon existing and validated building blocks. We identified IR techniques, as they are applied in traceability recovery [Borg et al., 2014] in general and feature location [Dit et al., 2011b] in particular, as part of a potential solution and performed a series of experiments with the goal to evaluate whether the overall approach is feasible. We illustrate the design and evolution of the experiments, pointing out practical implementation challenges that we encountered while adapting IR techniques proposed by research. Even though this research was driven by eventually developing a workable solution for test case selection decision support, we focus in this chapter on the lessons learned from designing and executing the experiments. Thereby, we make the following contributions:

- A novel application of IR techniques to support test case selection in a Software Product Line context.
- An illustration of three experimental setups, each leading to a set of new research challenges for IR applications on industry-grade data.

The remainder of this chapter is structured as follows. In Section 2 we present the context in which we conducted the research, illustrate the envisioned solution and discuss related work. Section 3 provides the formal problem definition that guides the overall design of the experiment. In Section 4 we illustrate the design, execution, results and lessons learned from each of the three experimental setups. The chapter concludes in Section 5, pointing out avenues for future work.

## 2 BACKGROUND AND RELATED WORK

This research is conducted in collaboration with ES (“Embedded Systems”, name anonymized for confidentiality reasons) and the solution development is driven by their particular context and requirements. In this section, we first give an overview of the case context and problem (Section 2.1). We take then a step back and analyze the problem with respect to the state of art (Section 2.2) before sketching a solution in Section 2.3. We discuss related work in Section 2.4.

### 2.1 Case Context

ES, developing both hard- and software for their worldwide marketed products, has a three-decade history in developing embedded systems, although the particular applications areas have changed over time.

#### 2.1.1 Variability Management and Consequences for Quality Assurance

In autumn 2011, we performed a lightweight process assessment [Pettersson et al., 2008] at ES in order to identify improvement opportunities, in particular in the coordination between requirements engineering and software testing. We interviewed 16 employees (line managers, product and project managers, test and technology leads, and test engineers) and reviewed project documentation, test runs and trouble reports of two recently completed projects. As observed by Thörn [2010] in small and medium sized companies, we identified also at ES a lack of variability management in the problem space (requirements analysis, specification and maintenance). This has historical reasons, mostly attributed to the strong technical support for managing variants in the solution space (separate development of a platform and product specific software). ES generally develops a new product generation on the basis of their current products, reusing and extending the existing requirements specifications. This has the advantage that the requirements management process is lightweight and requires minimal documentation. However, this can also lead to challenges for impact analysis of new features since relationships between requirements are not documented [Graaf et al., 2003]. The interviewed test engineers at ES also indicated that without variability management, they may select system test cases that are not applicable for a particular product. This can lead to a lower efficiency in test execution as information on applicability needs to be retrieved from the project (project manager or product expert) or by an in-depth analysis of the particular test case and product. Looking at the test case database at ES, for a typical product release, 400–1000 system test-cases are selected, run and evaluated, each consisting of

up to 20 manual test steps. An improved test case selection can therefore reduce rework and time-to-market.

ES initiated different programs to improve their quality assurance efficiency. They introduced a risk-based strategy in order to focus the testing effort on those parts of the product carrying a risk to contain faults, optimizing thereby available time and resources. The risk-based test selection is based upon expert judgments from technology leads, product experts and test maintainers, but also on historical data from test runs of similar products. Furthermore, automated integration tests are run on every version control system commit and before the product is handed over to the quality assurance (QA) department. A third avenue to achieve a more precise test case selection, developed at ES, is described next.

### *2.1.2 Test Case Selection Based on Product Configurations*

For ES, an important criterion to adopt any test case selection support is to reuse as many existing resources and artifacts as possible, causing very little additional up-front investments to the overall development process. The developed approach leverages on the existing technical infrastructure managing product variants in the solution space, where a central configuration file determines the activation state of a feature in the product. The approach consists of:

- a parser that determines the activation state of a feature from the central configuration file,
- the creation of a mapping between features and test-case modules (containing a set of related test-cases), verified by product experts.

A product delivered to QA can be analyzed automatically to determine the activation status of its features. The parser creates a Product Configuration Report (PCR) that is used by QA engineers to guide the selection of system test cases. Testers need thereby to map between product features and test modules. In the current implementation, this solution has the following weak points:

- The feature list in the PCR is incomplete, i. e. some test case modules can not mapped to a feature.
- Some test-cases need to be executed for every product. These test cases are also not mapped to a feature in the PCR.
- The granularity of the feature list in the PCR disallows a mapping to specific test cases.
- The mapping is static. If the organization of the test case modules changes or features in the central configuration file change, testers and product experts need to re-verify the mapping.

Based on these observations we started to explore alternative approaches to address the test case selection problem.

## 2.2 Overview of the State of Art

As illustrated in Section 2.1.1, ES resides in a Software Product Line (SPL) [Clements and Northrop, 2001] context, where verification and validation is difficult due to the large number of product combinations that can be created [Perrouin et al., 2010]. This problem is aggravated when the product is a combination of software and hardware, since late identification of defects when verifying an embedded system is costly [Broy, 2006, Ebert and Jones, 2009]. Model-based testing (MBT) techniques [Uutting et al., 2012] have been proposed to address this complex challenge from an SPL perspective (see Engström and Runeson [2011] for a review of solution proposals). However, many of these proposals assume that companies apply variability management [Babar et al., 2010] in their product lines, allowing the derivation of test strategies and the efficient selection of test cases. Chen and Babar [2011] reviewed the variability management techniques proposed in literature, showing that the majority is not validated in an industrial context. This is supported by the observation by Thörn [2010] that companies have control over variation in the solution space, lack however techniques and methods for variability management in the product and problem space. With a lack of variability management in the problem space, as it is the case for our case company, software product line testing principles [Pohl and Metzger, 2006] or test case selection techniques [Lee et al., 2012] are not applicable without upfront investment. Since reuse of existing resources and artifacts is a major criterion for ES to adapt a solution (see Section 2.1.2), we sought for alternatives.

Test case selection, in the context of regression testing, encompasses the problem of choosing a subset of test cases, relevant to a change in the system under test, from a set of existing test cases [Rothermel and Harold, 1996]. To address this engineering problem, various techniques have been developed (see Engström et al. [2010] and Yoo and Harman [2012] for comprehensive reviews). In general, they can be classified into white and black-box selection techniques. The former rely upon structural knowledge of the system under test, exploiting, for example, changes in source code, data and control flow, or execution traces [Yoo and Harman, 2012]. The latter use other work products, such as design documentation, to perform impact analysis [Yoo and Harman, 2012]. Both white and black-box test case selection techniques assume some sort of traceability between the changed artifact and the relevant test cases. Since the lack of variability management excludes the black-box selection approaches, ES has implemented traceability between product configurations and test cases as described in Section 2.1.2, with the observed drawbacks. Next, we give a motivation and outline for the solution we aim to implement and evaluate in the remainder of this chapter.

Table 6.1: Candidate artifacts for automated feature identification

| <i>Artifact</i>   | <i>Benefits</i>  | <i>Liabilities</i>   |
|---|--|--|
| <i>PFD</i> is a design document written in natural language that details of what a certain functionality consists of and how it is intended to be used. | Detailed and accurately linked to a particular product variant.  | Exists only for newly introduced features, i.e. is incomplete w.r.t. the overall functionality of a product variant. |
| <i>PCS</i> contains all available configuration options and dependencies among them, used to configure a product at build-time                          | Straightforward identification of activated/deactivated features   | Not all tested features are activated / deactivated at build-time  |
| <i>Source code</i>  | Complete w.r.t. identifying both build-time and run-time bound functionality.  | Lowest possible abstraction level for representing features  |
| <i>Version Control</i> data   | Provides semantic information to source code changes, such as commit comments, time, scale and frequency of changes. | Except for commit comments, little use for feature identification if not used in combination with source code.       |

### 2.3 Solution Development

We outline the solution development approach by breaking down the test case selection problem into two objectives.

The first objective is to identify the to-be-tested features in a particular product variant that is delivered to QA. Currently, product variants are not managed in the problem space (e.g. in the requirements specifications) at ES. Hence, other work products that are created during the project and that are up-to-date when the product is delivered to QA need to be used instead. Table 6.1 summarizes those candidate artifacts that fulfill these criteria.

The second objective is to map system test-cases to identified features. A system test-case contains information on the overall objectives of the test, acceptance criteria, and test steps. By establishing a trace link between a feature in a particular product variant and the set of relevant test-cases for that feature, test-case selection can be performed. In ES's solution, the

mapping between the PCR and test-case modules is performed manually, which leads to the drawbacks illustrated in Section 2.1.2.

Looking at the available artifacts in Table 6.1, only the source code artifacts provide complete information on the functionality in a particular product variant. On the other hand, the source code information (comments, method and variable identifiers, and literals) regarding each feature is given at a low abstraction level, compared to the other work products, i. e. the PCS and the PFD.

Inspired by ES's semi-automatic solution that uses the PCS and expert judgment for test-case selection, we envision a solution that is applicable on source code, providing feature existence information, and is not dependent on regular expert input. In the area of source code mining, in particular textual feature location techniques based on natural language processing (NLP)<sup>1</sup> or information retrieval (IR) [Grossman and Frieder, 2004] seem promising. The goal of feature location is to support software developers in maintenance tasks by identifying relevant locations in the source code, e. g. to remove a fault or to extend a feature. The premise of textual feature location is that comments, identifiers, and literals encode domain knowledge, representing features than can be located in the source code by a set of similar terms [Dit et al., 2011b].

The test case selection problem and the two outlined objectives at the beginning of this subsection can be formulated as a feature location problem: Given a test-case, representing a particular feature, locate the source code that implements that feature in the given product variant. The test case serves thereby as query, returning a list of artifacts ranked according to their similarity to the test case. Note that the specific problem of feature location can be also seen in the wider context of traceability recovery. The decision whether the given test case is selected or not, depends on whether the highest ranked artifact has a similarity score above or below a threshold  $\alpha$ . The particular value for  $\alpha$  is not known in advance and must be determined experimentally, with a confidence interval such that the test engineer can gauge the robustness of the suggestion. Determining  $\alpha$  can be achieved by sampling a set of features for which the following is known: (a) the test case(s) verifying the feature and (b) the source code implementing the feature. We call this relationship between feature, test case(s) and source code a feature chain. Once the value and confidence interval for the threshold  $\alpha$  is determined, it can be used to perform test case selection.

Note that this solution description is the starting point and not the outcome of our investigation. We provide a more formal problem definition in Section 3 and illustrate three experimental setups (Section 4) that were

---

<sup>1</sup> Note that the terms NLP and IR are independently used in literature to describe computerized processing of text and there is an overlap on what they comprise [Falessi et al., 2013]. In the remainder of the chapter we use exclusively the term IR to maintain consistency.

designed to study to what degree IR techniques can be used to support test case selection with industry grade data.

## 2.4 Related Work

### 2.4.1 Traceability Recovery

Mapping product features from source code to the respective test cases, as described in Section 2.3, is related to traceability [Gotel and Finkelstein, 1994] in general, and to the concept assignment problem [Biggerstaff et al., 1993] in particular. Concept assignment is the process of mapping concepts from the problem domain (e.g. a requirement or feature expressed in potentially ambiguous and imprecise natural language) to the solution domain (e.g. an algorithm expressed in precise and unambiguous numerical computations). Recovering traceability links between source code and natural language documentation using IR techniques was pioneered by Antoniol et al. [1999, 2000], Maletic and Valluri [1999] and Maletic and Marcus [2000, 2001]. These early studies envisioned the potential of IR techniques to support software engineers in program comprehension [Maletic and Valluri, 1999, Antoniol et al., 1999], requirement tracing and impact analysis, software reuse and maintenance [Antoniol et al., 1999]. Following these initial investigations, comprehensive experiments were conducted, studying particular IR techniques in depth (e.g. Marcus et al. [2004], Zhao et al. [2006], De Lucia et al. [2007], Poshyvanyk et al. [2012]) or comparing newly proposed techniques to previously studied ones (e.g. Marcus and Maletic [2003], Poshyvanyk et al. [2006], Cleary et al. [2009]). A large part of these studies use small scale data sets, derived from open source applications or public repositories. Furthermore, while a large body of work w.r.t. traceability recovery has been reviewed by Borg et al. [2014], only a few addressed the recovery of links between source code and test cases (e.g. De Lucia et al. [2006, 2009, 2011]), and have been evaluated only on student projects [Borg et al., 2014].

Besides the study of individual IR techniques and their traceability recovery performance, investigations into the hybrid techniques show promise. Gethers et al. [2011] combined deterministic and probabilistic IR techniques, exploiting the fact that they capture different information. While this hybrid traceability recovery approach outperformed the individual techniques, determining the number of topics in the probabilistic approach [Steyvers and Griffiths, 2007] and finding an optimal combination of the individual approaches are still research challenges [Gethers et al., 2011]. More recently, Qusef et al. [2014] combined textual and runtime information to recover traceability between source code and unit test cases. In their approach, first the runtime information is used to create a candidate list of traces, which is then further refined in a second step

which analyzes textual information. While the evaluation of this hybrid technique indicates that it outperforms the precision of individual techniques [Qusef et al., 2014], it requires the collection of a test execution trace and has been designed for unit-testing, i. e. white-box testing.

#### 2.4.2 Test Case Prioritization

While test case selection encompasses the problem of identifying test cases relevant to a change in the system under test, prioritization aims at ordering test cases such that defects are detected as early as possible [Yoo and Harman, 2012]. Islam et al. [2012] recover traceability links between source code and system requirements with IR techniques and use this information together with code coverage and test execution cost to identify optimal test orderings. This approach requires structural knowledge of the system under test, i. e. source code in this particular case. Thomas et al. [2014] propose therefore an approach that relies only upon data originating from the test cases. They use an IR technique to measure the similarity between test cases and prioritize them such that dissimilar test cases are run first.

#### 2.4.3 Configuration of IR Techniques

Numerous studies proposed and evaluated IR techniques for different software engineering tasks (see the systematic literature reviews by Dit et al. [2011b] and Borg et al. [2014]), leading to the need of systematically comparing and evaluating these techniques. The fundamental idea behind any configuration optimization is to guide the process by a data-driven parameter selection. For example, Falessi et al. [2013] use IR techniques to detect equivalent requirements and evaluate the performance of 242 configurations. While they identify an optimal technique, they point out that the particular parameters are dependent on the underlying data set, which means that the identification of the optimal technique configuration requires considerable effort. Lohar et al. [2013] propose therefore to use Genetic Algorithms [Goldberg, 1989] to search for the optimal IR technique configuration. A similar approach was proposed by Panichella et al. [2013] to configure probabilistic IR techniques (Latent Dirichlet Allocation [Blei et al., 2003]).

#### 2.4.4 Contribution

Traceability recovery supported by IR techniques has seen a lot of research in many application areas in software engineering (see Section 2.4.1). However, research on traceability recovery with IR techniques between source code and test cases has been rare [Borg et al., 2014], or not designed for

black-box system testing (e.g. Qusef et al. [2014]). In this chapter we propose to use IR techniques for test case selection in the context of Software Product Lines. While IR techniques have been applied for test case prioritization (see Section 2.4.2), the use in test case selection is, to the best of our knowledge, novel. IR techniques need to be configured in order to reach optimal performance. Recent studies show that configuring IR techniques should be driven by experiments since the performance depends on the underlying data (see Section 2.4.3). We use an industrial data set that is at least by an order of magnitude larger than in previous studies attempting to configure IR techniques. We report on the practical implementation of our proposed test selection techniques, the challenges we encountered and the lessons learned in evaluating the approach experimentally.

### 3 RESEARCH METHOD

In Section 2.3 we stated the informal hypothesis that textual feature location based on IR techniques can be used to support test case selection decisions. Therefore, using the goal-question-metric method [Basilis and Caldiera, 1995] to specify the goal [Dybå et al., 2005] of this study, we:

*Analyze IR techniques, for the purpose of evaluation, with respect to their support for test case selection, from the point of view of the practitioner, in the context of industry-grade software development artifacts.*

Based on this goal definition, we state the following research questions.

RQ-1 To what extent can state-of-the-art IR techniques be applied in a large-scale industrial context?

RQ-2 To what extent do the used software development artifacts influence the performance of IR techniques?

RQ-3 To what extent can the studied IR techniques support test case selection?

We defined the problem in the context of our industrial partner and use their data to design, implement and evaluate a solution candidate. This approach differs from previous IR experiments on textual feature location which are mostly limited to preliminary evaluations in academic contexts [Dit et al., 2011b]. Hence, the purpose of RQ1 is to understand the challenges and to identify solutions for conducting IR experiments on data sets whose characteristics correspond to data sets from the problem domain. We conduct an experiment that evolves over three setups and address RQ-1 by reporting on lessons learned and analyzing them w.r.t. previous IR experiments reported in literature. With RQ-2 we aim to understand the impact of input data characteristics on the performance of IR techniques. With IR performance we refer to both the techniques' efficiency in terms of computational cost and to the techniques' effectiveness (the specific effectiveness measures differ in the experimental setups and are defined there).

We address RQ-2 in Experimental Setup 2 and 3. The purpose of RQ-3 is to compare the studied IR techniques and to determine whether the approach of using feature location as an aid to test case selection is feasible. We address RQ-3 in Experimental Setup 3. The remainder of this section describes the overall design of our experiment.

### 3.1 Description of Objects

The objects of the experiment are software development artifacts belonging to the implementation (source code, configuration files, user help documentation) and the system test (natural language test steps) domain.

In the implementation domain, there is a set of features  $F = \{F_1, \dots, F_{nf}\}$ , where  $nf$  is the total number of features, that represent the configurable functionality of a family of systems  $V$ . The feature vector  $A_i = [a_1, \dots, a_{nf}]$ , where  $a_j = \{0, 1\}$ , identifies a particular variant  $V(A_i)$  of the system family  $V$ . Furthermore, there is a set of source code files  $C = \{C_1, \dots, C_{nc}\}$ , where  $nc$  is the total number of source code files, each consisting of a set of source code lines  $L_{Cj} = \{L_1, \dots, L_{nl}\}$ , where  $nl$  is the total number of lines in that source code file. Depending on the particular implementation of a feature, there exists a mapping of feature  $F_i$  to either:

- $C(F_i) \subset C$ , i. e. a subset of the source code files
- $L_{Cj}(F_i) \subset L_{Cj}$ , i. e. a subset of the source code lines
- a combination of the above, i. e. feature  $F_i$  is mapped to a subset of the source code files *and* a subset of source code lines

Note that  $C(F) \neq C$  and  $L_{Cj}(F) \neq L_{Cj}$ , i. e. there is a subset of source code files/lines that are not mapped to any configurable feature. Those source code files/lines are common to every system variant in  $V$ .

In the system test domain, there is a set of test cases  $T = \{T_1, \dots, T_{nt}\}$ , where  $nt$  is the total number of test cases. A test case is a natural language document that contains categorization information (test case name, test module name, test module information) and test execution information (assumptions, initialization information, acceptance criteria, test objective, test steps).

### 3.2 Feature Chains

Specific instances of the objects defined in Section 3.1 determine a *feature chain*, connecting features, source code and test cases (Figure 6.1). A feature chain consists of two links:

1. Feature to test cases: established by senior test engineers, linking a subset of all features  $F$  to a subset of all test-cases  $T$ , i. e.  $T(F_{1..n} \subset F) \subset T$ , where  $F_i \rightarrow T_{1..m}$ .

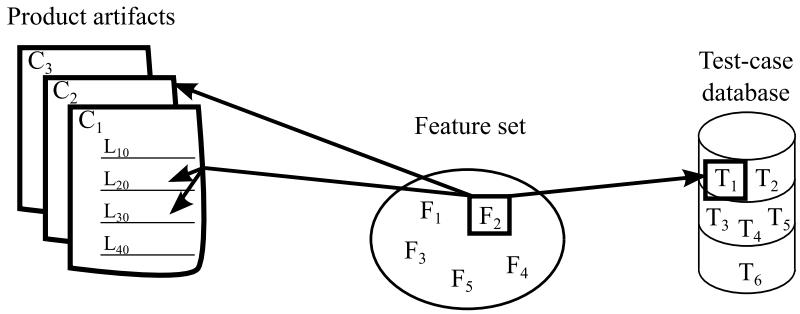


Figure 6.1: Example of a chain connecting a feature with test case and product artifacts

2. Feature to product artifacts: established by identifying the impact of the configuration switch activating/deactivating a feature on product artifacts (i. e. file and/or line addition/removal).

A feature chain  $I_i = \{F_i, C(F_i), L_{C_j}(F_i), T(F_i)\}$  connects the artifacts from the implementation and system test domain. Figure 6.1 illustrates this concept. Feature F<sub>2</sub> is verified by test case T<sub>1</sub>. This connection is established by a senior test engineer. Feature F<sub>2</sub> is implemented in source code file C<sub>2</sub> and in lines L<sub>20</sub> to L<sub>30</sub> in file C<sub>1</sub>. This connection is established by identifying the impact of the configuration switch activating/deactivating feature F<sub>2</sub> on C<sub>1</sub>, C<sub>2</sub>, and C<sub>3</sub>.

### 3.3 Independent Variables

The potential independent variables, i. e. the variables that can be controlled and manipulated in the experiment, are numerous. Since the overall architecture of IR techniques is that of a pipeline, individual components can generally be replaced as long as input/output constraints are satisfied. Falessi et al. [2013] propose a classification of IR techniques consisting of four main components: IR model, term extraction, weighting schema, and similarity metric<sup>2</sup>. For each of these dimensions, it is possible to identify multiple factors that deserve consideration in an experiment evaluating IR techniques. However, since the main purpose of this study is to determine the feasibility of IR techniques for test case selection, we limit the number of independent variables and values to a realizable number (see Table 6.2). As it turned out during the execution of the experiment, with increasing complexity and amount of data on which the IR techniques are applied, we had to remove values as they were not feasible to manipulate, but also

---

<sup>2</sup> Grossman and Frieder [2004] provides a broader overview

Table 6.2: Selected independent variables (IV) and values

| IV            | <i>Test case content</i>          | <i>Term extraction</i> | <i>IR Model</i> |
|---------------|-----------------------------------|------------------------|-----------------|
| <i>Values</i> | Full test case                    | Tokenization           | VSM             |
|               | Test case except test steps       | Stop words             | LSA             |
|               | Acceptance criteria and objective | Stemming               |                 |

*added* values as the data required e. g. additional pre-processing. These decisions are motivated and documented in the corresponding experimental setup in Section 4. We describe now the independent variables shown in Table 6.2 in more detail.

As described in Section 3.1, test cases consist of categorization and execution information, containing information that is not necessarily connected to the to-be-tested feature. For example, assumptions, initialization information and test steps may refer to pre-requisite functionality that is verified by another dedicated test case. Hence, it is necessary to identify the test case content that is the most effective, i. e. encode the most information on a feature with the least amount of noise.

Term extraction refers to pre-processing techniques applied incrementally on the analyzed texts, where the simplest form, tokenization and stop-word removal, can be extended by stemming [Falessi et al., 2013]. Furthermore, techniques exist that aim specifically to improve IR applied on source code, e. g. by splitting [Enslen et al., 2009, Dit et al., 2011a] or expanding identifiers [Hill et al., 2008, Lawrie and Binkley, 2011] or a combination of both [Guerrouj et al., 2011, Corazza et al., 2012]. Even though these techniques seem to improve the performance of IR, they have been implemented for and validated against only a subset (Java, C, C++) of the programming languages we encountered in a product (see Section 4.1.3). Therefore, we decided against adding them as a manipulated variable in our experiment.

An IR model evaluates the semantic similarity between text documents [Falessi et al., 2013]. The vector space model (VSM) [Salton et al., 1975] and latent semantic analysis (LSA) [Deerwester et al., 1990] both represent documents as term-frequency vectors and consider the distance between these vectors as semantic dissimilarity. LSA, an extension to the VSM, considers co-occurrence of terms [Falessi et al., 2013] in order to address synonymy (the same meaning – different terms) and polysemy (different meanings – the same term) [Deerwester et al., 1990]. Note that LSA is a parameterized model, increasing the number of independent variables.

### 3.4 Validity Threats

We use Wohlin et al. [2000] to structure this analysis and discuss the threat categories in the suggested priority for empirical software engineering experiments: internal, external, construct and conclusion threats. For each category, we discuss only those threats that apply to the experiment design formulated in this section.

#### 3.4.1 Internal Validity

The *instrumentation* threat refers to the effects caused by the artifacts used for experiment implementation. The impact of the used artifacts is a central aspect of the study overall, but in particular of the experimental design and its implementation in three setups. As such, we are explicitly addressing and discussing instrumentation and its effect in each setup.

The *selection* threat refers to the variation of the chosen objects in the experiment. The feature chains are determined by the possibility to establish an association between a particular feature and the test cases that verify that feature. As such, the selection of features is biased towards those for which the tasked test engineers could create such a mapping. We have however no indication that the non-random selection of features introduced a systematic bias towards product variants and their artifacts or types of system test cases.

#### 3.4.2 External Validity

The *interaction of selection/setting and treatment* threat refers to the selection of objects from a population that is not representative for the population to which we want to generalize. We address this threat by sampling objects from an industry setting which is the population to which we generalize our results.

#### 3.4.3 Construct Validity

*Inadequate pre-operational explication of constructs* refers to the threat of insufficiently defining the theoretical constructs of the experiment. We address this threat by formalizing the relationships of the objects, i.e. the feature chain and its components, in the experimental design and using this formalization in the execution of the three experiments. However, there is a threat that the mapping of test cases to features, necessary for constructing a feature chain, was biased by the understanding and experience of a single tester. We addressed this by involving two test engineers in this process.

A *mono-operation bias* exists in an experiment if the limited number of independent variables, objects or treatments leads to an incomplete picture

of the theory. While we aimed at selecting a sensible variation of independent variables, the choice was eventually limited by the practical constraints of implementing the experiment on industry grade data sets, that addressed external validity threats. This is a trade-off between conflicting validity threats one has to consider when performing applied research [Wohlin et al. \[2000\]](#). Other studies focus on a broader coverage of IR configurations, e. g. [Falessi et al. \[2013\]](#) and [Biggers et al. \[2014\]](#).

#### 3.4.4 Conclusion Validity

There is a moderate threat of *low statistical power* w.r.t. the conclusions made in Experimental Setup 3. The cost for establishing an observation (feature chain) considerably limited the number of observations that could be made with reasonable effort, also from the industry participants.

We address the threat of *violated assumptions of statistical tests* by an in-depth analysis of the experimental design and choice of statistical techniques (see Section 4.3.5).

While the implementation of corpus creation, term extraction, similarity analysis and evaluation differed in the three experimental setups (see Figure 6.2) due do their varying objectives, we consider the threat of *reliability of treatment implementation* low. The implementation of the four steps did not change within a experimental setup and is available in the supplementary material [[Unterkalmsteiner et al., 2014b](#)] for reference.

## 4 EXPERIMENTAL SETUPS AND EVALUATION

In this section we illustrate three major experimental setups whose implementation is guided by the design presented in Section 3. Each setup had a particular objective and its implementation lead to lessons learned, motivating and shaping the objectives of the subsequent setup.

The execution of the experiment follows the general flow shown in Figure 6.2 in all three setups. In Step 1, we create the corpus for a chain, consisting of the configured product artifacts and test-cases. In Step 2, the corpus is textually pre-processed, which removes noise and prepares the data for the subsequent analysis in Step 3. We perform the evaluation of the IR technique in Step 4. With each experimental setup, the steps are refined to address issues and lessons learned from the predecessor. An overview of the differences between the three experimental setups is shown in Figure 6.2, while the details are discussed in the respective subsections.

|                                       | Step 1<br>Corpus creation   | Step 2<br>Term extraction  | Step 3<br>Similarity analysis                  | Step 4<br>Evaluation     |
|---------------------------------------|---|--|--|--------------------------|
| Experimental Setup 1<br>(Section 4.1) | 1 Chain<br>Manual configuration<br>Subset of product<br>3 test-case variants        | Remove special chars<br>Filter stopwords<br>Stemming                       | Vector Space Model<br>Latent Semantic Analysis | Difference of similarity |
| Experimental Setup 2<br>(Section 4.2) | 6 Chains<br>Manual configuration<br>Complete product<br>3 test-case variants        | Remove special chars<br>Filter stopwords<br>File filter<br>Fact extraction | Vector Space Model<br>Parallelization          | Difference of similarity |
| Experimental Setup 3<br>(Section 4.3) | 10 Chains<br>Automatic configuration<br>Complete product<br>4800 test-case variants | Remove special chars<br>Filter stopwords<br>File filter<br>Fact extraction | Vector Space Model<br>Parallelization          | Test-case ranking        |

Figure 6.2: General experiment procedure (horizontal axis) and changes between the three experimental setups (vertical axis)

#### 4.1 Experimental Setup 1: Pilot Experiment

The motivation for preparing a pilot experiment is to get an intuition whether IR techniques can be used to solve the test case selection problem. Therefore, we designed an experimental setup that tests the sensitivity of IR. In other words, we want to determine whether the signal provided by the test cases is strong enough to differentiate between a feature-activated and a feature-deactivated corpus.

Formally, using the notation introduced in Section 3.1, we have in Experimental Setup 1 one feature chain  $I_1$  that includes feature  $F_1$  and test case  $T_1$ . Therefore, we have two feature vectors  $A_a = [1, a_2, a_3, \dots, a_{fn}]$  and  $A_d = [0, a_2, a_3, \dots, a_{fn}]$ , resulting in two system variants  $V(A_a) = \{C_1, \dots, C_{cn}\}$  and  $V(A_d) = \{C_1, \dots, C_{cm}\}$ . We evaluate whether  $\text{sim}(T_1, V(A_a)) > \text{sim}(T_1, V(A_d))$ , where  $T_1$  is the test case mapped to feature  $F_1$ . As for independent variables, we manipulate the content of  $T_1$  and the IR model  $\text{sim}$  (see Table 6.2). Figure 6.2 illustrates the configuration of each step in this setup, further detailed in subsections 4.1.1 – 4.1.4 and in Figure 6.3.

The overall objectives of the pilot experiment are to:

- increase the understanding of the analyzed data (product artifacts, test cases)
- identify and evaluate IR tools which can be used to implement and execute the experiments
- provide evidence on whether this approach is generally feasible or not

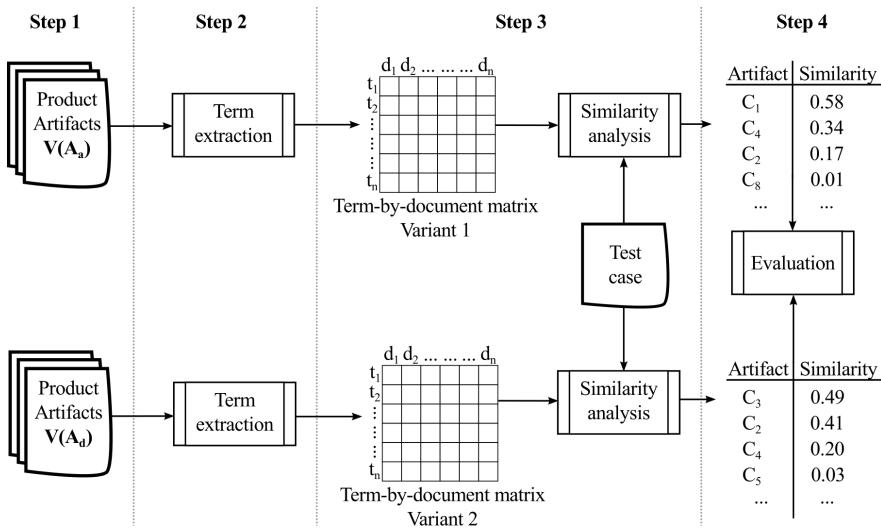


Figure 6.3: Pilot setup details

We discuss in Sections 4.1.4 and 4.1.5 to what extent these objectives have been achieved.

#### 4.1.1 Step 1 – Corpus Creation

The product is configured by a collection of GNU autotools [Calcote, 2010] scripts and built with GNU make [Feldman, 1979]. In a typical product build, several hundred software packages are fetched from the version control system server, configured, compiled and packaged into an image that can be installed onto the respective camera model.

For the pilot experiment, we selected one feature that is implemented in a relatively small software package (368 files). On activation, the feature affects 10 source code files (8 modifications and 2 additions). The build system uses filepp<sup>3</sup>, a file pre-processor, to apply configuration switches on plain text and HTML files. By manually invoking filepp on the software package that implements the user interface, we generated two variants with 10 distinct and 358 shared files.

#### 4.1.2 Step 2 – Term Extraction

We applied the following pre-processing steps on both the source code and test case variants:

<sup>3</sup> <http://www-users.york.ac.uk/~dm26/filepp>

- filtered numeric, punctuation and special characters<sup>4</sup>
- filtered stop-words; English, and HTML/javascript keywords
- performed word stemming

Stop-word filtering and word stemming were implemented using the generic data mining tool Rapidminer<sup>5</sup>, which was also used in the next step.

#### 4.1.3 Step 3 – Similarity Analysis

Figure 6.3 illustrates the pilot setup in more detail, emphasizing the fact that we are working with two corpora ( $V(A_a)$  and  $V(A_d)$ ) with activated and deactivated feature).

When creating the term-by-document matrix in Step 3, one has to decide on the document granularity, i. e. on how the corpus is divided into documents. Commonly, when analyzing similarity in source code, documents are defined as a method [Marcus et al., 2004, Cleary et al., 2009], a class [Antoniol et al., 2002] or a file [Marcus and Maletic, 2003]. We defined a file as a document due to the heterogeneity of the source code (markup, procedural, object oriented languages).

We applied VSM and LSA on the corpora, resulting in a list of ranked documents according to their similarity to the test case. Figure 6.4 shows the results of 12 runs of the pilot experiment: with the VSM, we varied the test case content ( $V_1$ - $V_3$ ), whereas with LSA, we also varied dimension reduction ( $V_4abc$ - $V_6abc$ ). For each run, we plotted the cosine similarity of the first 30 ranked documents (for all runs, the similarity values approach values below 0.1 after 30 documents and showing all documents would render the plots more difficult to read). Furthermore, documents that differ between the two corpora, i. e. that are affected by activating/deactivating the feature, are marked with an “x”-symbol.

Looking at Figure 6.4, it can be observed that for all runs, in the feature activated corpus (solid line), configured documents are ranked first (e. g. 4 in  $V_1$ , 2 in  $V_4a$ , etc). This shows that, for this particular corpus and test case, all IR techniques and combinations are able to locate the feature in question. However, we are interested in differences between corpora (solid and dashed lines). These differences are visible in runs  $V_1$ - $V_3$  (VSM), i. e. the maximum similarity of the feature activated corpus is higher than the one of the feature deactivated corpus. Furthermore, the solid line (activated corpus) is consistently above the dashed line (deactivated corpus). This indicates that, at least visually, we can differentiate between the two corpora. This changes in runs  $V_4$ - $V_6$  (LSA), rendering the visual differentiation more difficult. We can observe that the maximum similarity in run

---

<sup>4</sup> using GNU sed, <http://www.gnu.org/software/sed/>

<sup>5</sup> <http://www.rapidminer.com>, version 5

V4a is higher in the feature deactivated corpus higher than in the feature activated corpus. Furthermore, in runs V(456)b and V(456)c the solid line intersects the dashed line at around file 8, indicating that the overall calculated similarity between test case and feature deactivated corpus is higher than for the feature activated corpus.

These results are promising as they indicate that a differentiation between a feature activated and deactivated corpus is possible, i. e. that there is a discerning signal in this particular test case and IR techniques are able to detect that signal. However, in order to determine which combination of test-case component and IR technique performs best, we need to define a measure that is able to express the visually determinable difference as a numerical value, as shown next.

#### 4.1.4 Step 4 – Evaluation

In Section 4.1 we defined an experimental setup with two corpora in order to test whether IR techniques can differentiate, given a test case, between the two product variants. This is motivated by our aim for test case selection, which needs to determine the existence/non-existence of a feature, but not necessarily its exact location.

However, this also means that the traditional class of performance measures, e. g. precision and recall, based on a confusion matrix, cannot be used for the following reason. In a binary classification problem, the confusion matrix contains four classes: true positives, true negatives, false positives, and false negatives. Given an arbitrary threshold value for the calculated similarity between document (source code) and query (test case), one can define a confusion matrix for the corpus where the feature is activated. However, given a corpus where the feature is deactivated, the number of true positives and false negatives, i. e. the number of relevant documents, is by definition zero. In other words, no classification takes place in the feature deactivated corpus. Hence, measures based on the confusion matrix are unsuitable in this case.

Therefore, we develop a measure based on similarity and the given ground truth. Similarity, a direct measure as opposed to the derived precision/recall pair, has been used by Falessi et al. [2013] to construct a credibility measure. Following this example, we construct a differential similarity measure,

$$\text{Dsim} = \max(\text{sim}_{\text{activated}|\text{configured}}) - \max(\text{sim}_{\text{deactivated}}) \quad (6.1)$$

which is the difference of the maximum similarity values of the two corpora. Note that the maximum similarity for the feature activated corpus is taken from the artifact that was actually affected by the feature activation, i. e. is configured. Since the calculated similarity between two documents ranges from 0 to 1, the range for Dsim is between -1 and 1. 1 indicates a

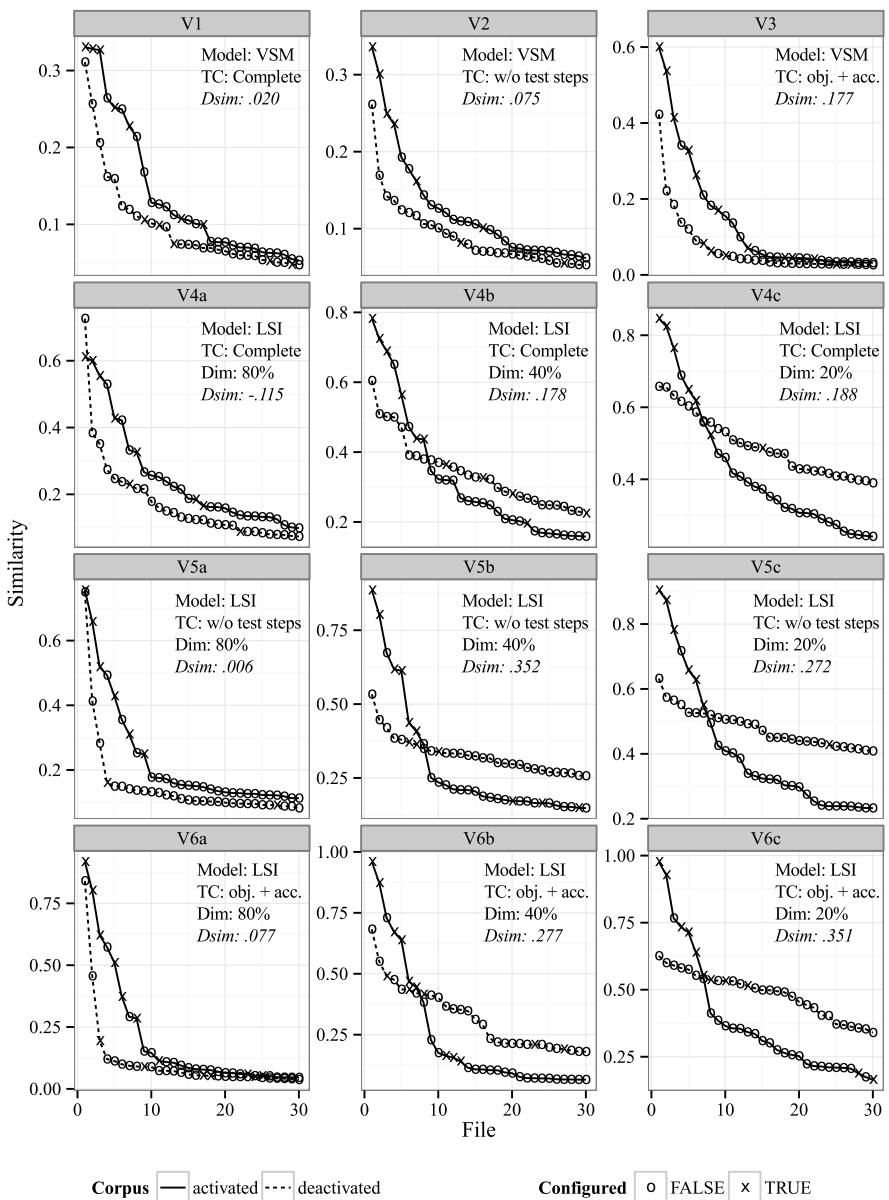


Figure 6.4: Pilot experiment results

perfect true differentiation, 0 no differentiation at all, and -1 a perfect false differentiation. A false differentiation occurs when the IR technique fails to connect the test case to the correct, i. e. configured, source code artifacts.

It would be possible to construct sophisticated measures, incorporating more of the distinguishing features of the curves in Figure 6.4 and the characteristics of feature locations. For example, one can assume that features are implemented in a small number of files out of a larger set. Then, the similarity for this small number of files would be large, followed by a drop-down and a quick convergence to 0 (as it can be observed in all curves of the feature activated corpus in Figure 6.4). For the feature deactivated corpus, one would expect only a small drop-down in similarity and a rather slow convergence to 0, as it can be observed in V(456)c in Figure 6.4. These features of the curves, drop-down and convergence of similarity measures, could be used to define more accurate differential similarity measures. However, this approach would be threatened by an over-fitting of the measure definition to the given data, resulting in a measure that would represent well the current, but not the future data. Hence, we choose the simple Dsim measure that relies only on the maximum similarity values.

#### 4.1.5 Discussion and Lessons Learned

We start this section by discussing the results of the evaluation and then elaborate on the lessons learned from the implementation of Experimental Setup 1, addressing RQ-1.

Looking at Figure 6.4, run V5b, V6c and V6b achieve the largest Dsim value, indicating that LSA performs better than VSM (V1–V3). In general, we can observe that the more specific the test case content, the larger the Dsim value, i. e. using the complete test case is consistently worse (for both VSM and LSA) than using a subset of the test case. This is an important result as it indicates that test cases contain noise that can be reduced by a selective removal of test-case content.

In this pilot setup, we test our idea of using IR techniques for test case selection. The results, illustrated in Figure 6.4, indicate that for this particular chain and subset of documents, we can indeed differentiate between a feature activated and a feature deactivated variant of a product, using the corresponding test case as a probe. However, further experimentation as described in Section 4.2 is required to investigate the scalability and generalizability of the approach.

Looking at the number of published studies in the field of feature location [Dit et al., 2011b], or in other areas where IR techniques are applied, e.g. traceability research [Borg et al., 2014], one would assume that literature provides both technical and methodological guidance to implement and evaluate IR-based solutions, adapted for the particular problem con-

text. However, very few studies provide enough detail to implement their proposed approach. None of the 14 publications that study textual feature location techniques based on VSM and/or LSA reviewed by [Dit et al. \[2011b\]](#) provides prototype implementations. Few mention existing tools that were used to implement parts of the proposed approaches. [Abadi et al. \[2008\]](#) and [Gay et al. \[2009\]](#) use Apache Lucene [[The Apache Software Foundation, 2014](#)] as the VSM implementation, and [Cleary et al. \[2009\]](#) mention SVDPACKC [[Berry, 2014](#)] as an implementation for singular value decomposition (SVD), a procedure required for LSA. [Lormans and van Deursen \[2006\]](#) use the Text to Matrix Generator toolbox [[Zeimpekis and Gallopoulos, 2006](#)], a Matlab module for text mining.

However, none of these tools lends itself to set up an experimental environment that allows one to explore data and technologies. Therefore, we implemented the process in Figure 6.2 with the off-the-shelf data-mining tool Rapidminer. The tool provides text processing capabilities and the necessary operators to implement VSM and LSA models. Furthermore, analyses can be constructed through the component-based graphical process designer, allowing for quick exploration of ideas. However, this convenience limits flexibility, e.g. by providing only a limited number of term weight functions<sup>6</sup>. Furthermore, the execution time for a single experiment run in the pilot does not scale to the amount of data in an actual chain. We implemented therefore the pilot process and the following experimental setups with shell scripts the statistical software package R [[Crawley, 2007](#)], available in the supplementary material [[Unterkalmsteiner et al., 2014b](#)]. While there exist efforts to reduce the barrier for conducting IR experiments, e.g. with TraceLab [[Cleland-Huang et al., 2011](#), [Dit et al., 2014](#)], researchers seem to require the flexibility of general purpose statistical software. For example, [Dit et al. \[2013\]](#) piloted their approach with TraceLab, implemented the final experiment [[Panichella et al., 2013](#)] however with R. This mirrors our experience with Rapidminer and R.

The studies reviewed by [Dit et al. \[2011b\]](#) and [Borg et al. \[2014\]](#) usually describe process steps, such as corpus creation, indexing, query formulation, and ranking. However, this information alone is insufficient to reimplement the proposed approaches as the IR techniques' behavior is defined by a set of parameters [[Thomas et al., 2013](#)], including pre-processing steps, input data, term weights, similarity metrics and algorithm-specific variations such as the dimensionality reduction factor  $k$  for LSA. Choosing  $k$  is a matter of experimenting with the data at hand to identify the value that provides the best retrieval performance [[Deerwester et al., 1990](#)]. When comparing different algorithms, it is necessary to identify the optimal parameters in order to provide a "fair" comparison [[Hooker, 1995](#)].

---

<sup>6</sup> Nevertheless, standard functionality can be extended through Rapidminers' plugin mechanism

This process is referred to as parameter tuning (see for example [Lavesson and Davidsson \[2006\]](#) and [Arcuri and Fraser \[2011\]](#)). Few of the studies reviewed by [Dit et al. \[2011b\]](#) and [Borg et al. \[2014\]](#) tune  $k$  to their LSA application. Some pick  $k$  based on experiences from previous studies (e.g. from [Deerwester et al. \[1990\]](#) and [Dumais \[1992\]](#)), which is however a questionable practice: these early studies use benchmark data-sets unrelated to the software engineering context. Furthermore, SVD on large, sparse matrices is a computationally expensive operation. Iterative implementations (e.g. [Berry \[2014\]](#)) using the Lanczos algorithm [[Cullum and Willoughby, 2002](#)] approximate the singular values, effectively reducing the runtime of LSA by limiting the number of dimensions. Computational cost should however not be a hidden driver for parameter selection as it biases the comparison [[Hooker, 1995](#)].

Few studies motivate, empirically or theoretically, their choice of  $k$ . A notable exception is [De Lucia et al. \[2007\]](#) where the performance of LSA is systematically evaluated. Since the document space in their experiment was small (150), it was computationally not costly to vary  $15 \leq k \leq 150$ . The authors observed that LSA performance, measured with precision and recall, did not vary much when  $k$  was set between 30%–100% of the document space, attributing this behavior to the small number of documents used in their experiments, and leading to the conclusion that  $k$  should be a user-configurable parameter in their proposed tool. [Poshyvanyk et al. \[2006\]](#) worked with a larger document space (68,190), varied  $k$  however only between 0.4%–2.2% ( $k = [300, 500, 750, 1500]$ ), motivating their decision that larger values for  $k$  are impractical to compute. They also observed that varying  $k$  did not significantly influence LSA performance.

However, concluding from these results that an optimal value for  $k$  should be between 30–500 would be wrong. [De Lucia et al. \[2007\]](#) observed a decline in performance when  $k$  was set to less than 10% of the document space. [Poshyvanyk et al. \[2006\]](#) never reached that percentage due to the computational cost involved in SVD. Not optimizing model parameters can lead to contradicting conclusions on which IR technique performs better, as the following example illustrates. [Oliveto et al. \[2010\]](#) compare, among other models, LSA with Latent Dirichlet Allocation (LDA). While  $k$  for LSA is not reported, the number of topics in LDA is varied between 50 and 300. The authors conclude that “the LDA-based traceability recovery technique provided lower accuracy as compared to other IR-based techniques” [[Oliveto et al., 2010](#)]. [Asuncion et al. \[2010\]](#), using the same standardized data set (EasyClinic<sup>7</sup>), parameterize LSA with  $k = 10$  and LDA with  $t = 10$ , and conclude that LDA performs better than LSA, i.e. the opposite of [Oliveto et al. \[2010\]](#). These contradicting results illustrate the importance of parameter optimization.

---

<sup>7</sup> <http://web.soccerlab.polymtl.ca/tefseo9/Challenge.htm>

We conclude that setting the dimensionality reduction factor for LSA is still an open issue (as initially observed by Deerwester et al. [1990]) and that studies that aim for a fair comparison between LSA and other techniques need to empirically determine optimal parameters for all compared models, based on the data at hand. In particular, a systematic evaluation of LSA on a large document space with  $k > 1,500$  is still required.

## 4.2 Experimental Setup 2: Scaling Up

While Experimental Setup 1 focused on studying the feasibility of using IR techniques for test case selection, this setup aims at evaluating the scalability of the approach. Concretely, the objectives are to study its behavior with respect to:

- computational scalability of the 4 steps (corpus creation, term extraction, similarity analysis and evaluation), and
- accuracy, i. e. a larger corpus means that the feature location task becomes more difficult since there are many more irrelevant than relevant documents

In this setup, we apply the piloted approach shown in Section 4.1 on a set of corpora created from the complete product. Formally, we have in Experimental Setup 2 six feature chains  $I_{1\dots 6}$  that include six features  $F_{1\dots 6}$  and six test cases  $F_{1\dots 6}$ . Therefore, we have 12 feature vectors  $A_{a1\dots 6}$  and  $A_{d1\dots 6}$  resulting in 12 system variants  $V(A_{a1\dots 6})$  and  $V(A_{d1\dots 6})$ . We evaluate whether  $\text{sim}(T_{1\dots 6}, V(A_{a1\dots 6})) > \text{sim}(T_{1\dots 6}, V(A_{d1\dots 6}))$ . As for independent variables, we manipulate the content of  $T_{1\dots 6}$ . The characteristics of this setup are summarized in Figure 6.2 and detailed in subsections 4.2.1–4.2.4. We report on the lessons learned in Section 4.2.5.

### 4.2.1 Step 1 – Corpus Creation

We selected six features, resulting in chains consisting of between 67,238 and 67,334 files, depending on the particular feature activation state. As in Experimental Setup 1, we created the different product configurations manually. However, in contrast to the limited set of files used in Experimental Setup 1, the complete product contains source code written in various programming languages, and build, configuration and documentation files. Furthermore, we had to consider more configuration mechanisms. In Experimental Setup 1, the feature activation/deactivation could be performed with filepp alone since only a subset of the product (the one configured with filepp) was considered for the corpus. However in this setup, GNU autotools mechanisms (i. e. depending on configuration, adding or removing files to the build process) and pre-processor mechanisms from the GNU compiler tool-chain needed to be considered in addition. In this

setup we did not build the product (this will be explored in Experimental Setup 3 in Section 4.3), but traced the impact of feature activation/deactivation and implemented it manually in the product artifacts.

For each feature chain, we randomly selected one test-case from the pool of test cases identified by the test engineers as relevant for the particular feature. As in the pilot experiment, we created three test case variants that differed by the amount of included information.

#### 4.2.2 Step 2 – Term Extraction

With the increase in size but also the variety of the corpus, term extraction became more challenging. We decided to eliminate term stemming as the processing time for one feature chain amounted to 20 hours. This likely increased the number of terms in the term-by-document matrix. On the other hand, we added a filter that removed binary files from the corpus and added a fact extraction process, inspired by Poshivanyk and Marcus [2007]. We used srcML [Maletic et al., 2002] to extract identifier names, comments and literals from C and C++ source code, reducing the amount of irrelevant terms in the corpus. Concretely, for the kernels' source code files, the term count could be reduced from 237,393 to 195,019 (a reduction of 18%) with fact extraction.

#### 4.2.3 Step 3 – Similarity Analysis

To implement the term-by-document matrix we used an R text-mining package [Feinerer et al., 2008], which provides efficient means to represent highly sparse matrices. This is required to be able to efficiently process a large corpus in memory. Furthermore, we improved the computational performance in two ways.

First, we optimized the calculation of similarity values by changing our initial, iterative, implementation into a vectorized form, exploiting the performance advantages of array programming [Iverson, 1980]. This reduced the similarity calculation using the VSM for one chain (corpus with 60,000 files) from 20 hours to 8 hours.

Second, we changed our implementation of VSM to support parallelization, allowing us to distribute the workload among several CPUs. We chose to use the snowfall package [Knaus, 2013] for this task as it allows to choose at run-time, without changing code, whether to run in cluster or in sequential mode, which is useful for testing the implementation. With the use of a cluster consisting of 8 cores, we could reduce the computation time for one chain from 8 to approximately 1 hour.

Table 6.3: VSM results using the Dsim measure

| <i>Variant / Chain</i> | 1     | 2      | 3      | 4      | 5      | 6      |
|------------------------|-------|--------|--------|--------|--------|--------|
| $V_1$                  | 0.120 | -0.406 | -0.139 | -0.490 | -0.259 | 0.004  |
| $V_2$                  | 0.170 | -0.355 | -0.141 | -0.504 | -0.244 | -0.009 |
| $V_3$                  | 0.228 | -0.486 | -0.215 | -0.517 | -0.205 | -0.010 |

#### 4.2.4 Step 4 – Evaluation

We used the Dsim measure, introduced in Section 4.1.4, to evaluate the performance of VSM to differentiate between the activated and deactivated corpus using a test-case. Table 6.3 shows the results of the 18 experiment runs: 6 chains and 3 test-case variants each. Recall that for a true differentiation, the Dsim measure ranges between 0 and 1.

Looking at Table 6.3, the results from chain 1 indicate that a true differentiation is possible. This confirms the results from Experimental Setup 1, which consisted of the same feature and test-case, however with a subset of the product artifacts. On the other hand, the Dsim measure of chains 2–6 indicates that the test-cases cannot differentiate between the feature activated and deactivated corpus. Looking at chain 2, shown in Figure 6.5a, we observe that there is no dissimilarity between the feature activated and deactivated corpus among the first 70 files. For chains 4 and 5 the curves look similar and are therefore not shown in Figure 6.5. Common to these three chains is the low number of files that are affected by a feature activation (between 2 and 7) and the change is minimal (typically a parameter replacement in a configuration file). In these cases, IR techniques are unlikely to work for test case selection as the changes between variants are minimal.

The situation is different for chains 3 and 6. Even though the Dsim measures in Table 6.3 indicate no differentiation, Figure 6.5b indicates a slight, and Figure 6.5c a more pronounced difference between activated and deactivated corpus. This suggests that the Dsim measure discards too much information by taking only the maximum similarity values into consideration.

#### 4.2.5 Discussion and Lessons Learned

We start this section by discussing implementation issues of Experimental Setup 2, and then elaborate on the lessons learned from the results and evaluation, addressing both RQ-1 and RQ-2.

The main concern in this setup is to scale the implementation of the experiment from a corpus with a few hundred files to one with 67000 files.

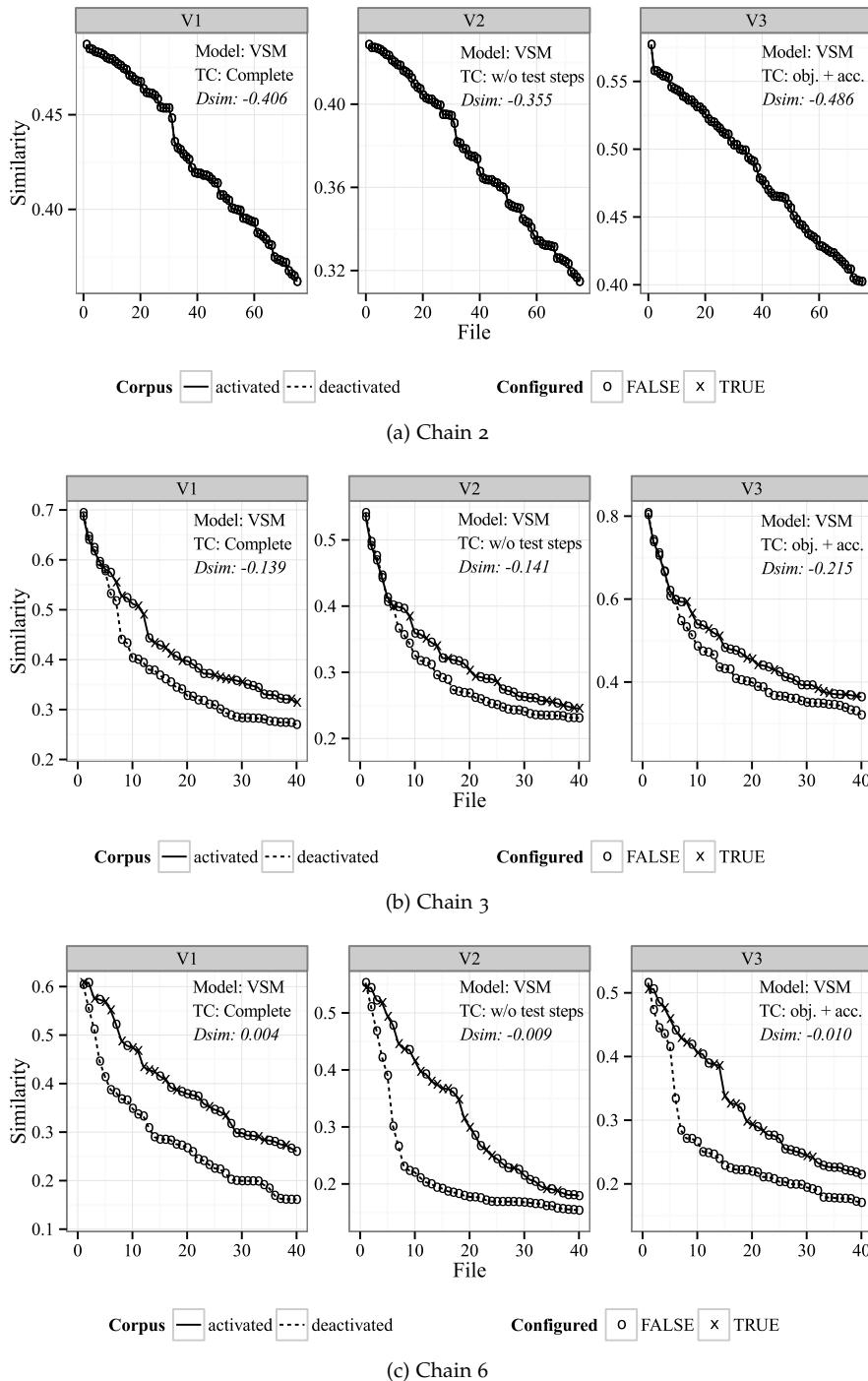


Figure 6.5: Results on three complete product variants

Table 6.4: Large scale IR experiments on source code

| <i>Publications</i>          | <i>IR model</i> | <i># Documents</i> | <i># Unique Terms</i> |
|------------------------------|-----------------|--------------------|-----------------------|
| Poshyvanyk et al. [2012]     | LSA             | 18,147             | 17,295                |
| Moreno et al. [2013]         | LSA/VSM         | 34,375             | Not stated            |
| Gay et al. [2009]            | VSM             | 74,996             | Not stated            |
| Poshyvanyk and Marcus [2007] | LSA             | 86,208             | 56,863                |
| Liu et al. [2007]            | LSA             | 89,000             | 56,861                |
| Experimental Setup 2         | VSM             | 67,334             | 359,954               |

After all, we are interested in studying the behavior of the IR techniques and solutions on a realistic data set that mirrors the characteristics of the target problem. The application of IR models (VSM and LSA) is thereby constrained by two factors: memory usage and computational time.

Since the term-by-document matrix is sparse, there exist efficient means to represent the matrix in memory, storing only non-zero values. However, this requires that matrix calculations support this format. Concretely, vectorized multiplication requires (by definition) that the multiplicands are stored as vectors. This is a requirement that we used in Step 3 (see Section 4.2.3) to our advantage. We created sub-matrices from the sparse term-by-document matrix that would fit in memory and distributed the cosine calculation for the VSM model among a cluster. Looking at the studies discussed in Section 4.1.5, only a few applied IR models on large corpora. Table 6.4 compares the average corpus in this study with the largest corpora identified in the relevant literature. None of these studies analyzes the implications of the large corpora on memory consumption and applicability of the proposed approaches in an actual solution that could be used in industry.

The second factor that constrains the applicability of the IR techniques is computational time. In Section 4.2.3, we have shown how the computational efficiency for the VSM model was increased by vectorization and parallelization of matrix multiplications. The LSA model requires a singular value decomposition of the term-by-document matrix. This operation is expensive with respect to execution time and difficult to parallelize for sparse matrices [Berry et al., 2006]. We ran a benchmark on a corpus with 35,387 documents and 193,861 terms, measuring the runtime of the SVD operation with various dimension reductions ( $k = 50, 100, 200, 300, 400, 500, 600, 700, 800$ ). We used the irlba package [Baglama and Reichel, 2014], which allows partial SVD calculations in contrast to the standard SVD implementation in R. Figure 6.6 shows the measured runtime in hours versus the number of dimensions. For example, the runtime

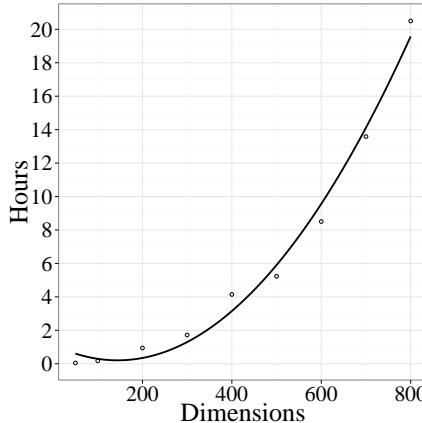


Figure 6.6: SVD runtime and dimensions

with  $k = 300$  amounts to 1 hour and 43 minutes, whereas with  $k = 800$  the SVD computation requires 20 hours and 30 minutes. We fitted a simple quadratic linear regression model to the data, explaining 99% of the observed variance (see Figure 6.6), which we could use to extrapolate the runtime for higher values of  $k$ .

In Experimental Setup 1, we varied  $k$  as a percentage of the number of documents in the corpus (80%, 40%, 20%). If we would apply the same strategy in this setup with 67,000 documents, even with only a 20% reduction ( $k = 13700$ ), the SVD runtime would amount to 8,287 hours, or 345 days. This illustrates why determining an optimal  $k$  is rather impractical and has been done only in a limited manner in the past (see discussion in Section 4.1.5). However, experimentation represents the only way to determine the optimal  $k$  for the data at hand [Deerwester et al., 1990]. Possible strategies that would make such experimentation possible, in particular with source code as documents, are to:

- Reduce the number of terms in the corpus: as we have shown in Section 4.2.2, fact extraction in C source code files can reduce the number of terms. This idea can be generalized to encompass other types of source code files.
- Exclude irrelevant documents from the corpus: we used every text-based document in the product repository as input. However, one could reduce this set to documents that are used in the product build process, effectively selecting only relevant documents as input. We explore this strategy in Experimental Setup 3 (Section 4.3).
- Use of parallel/distributed SVD implementations: we used a serial implementation which neither exploits multi-core processors nor can be run on a distributed system. Hence, one could explore solutions

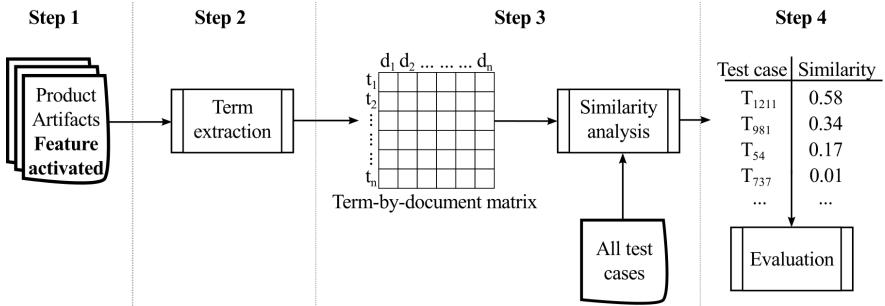


Figure 6.7: Ranking setup

that parallelize the SVD computation, e.g. pbdR [Ostrouchov et al., 2012], SLEPc [Hernandez et al., 2005] or gensim [Řehůřek, 2011]. Note that incremental SVD computations, as suggested by Brand [2006] and Jiang et al. [2008], would be of no benefit for model selection, since we are interested to vary  $k$  and typically do not update the corpus.

Due to the inefficiency of the SVD computation, we decided to exclude LSA from our further experimentation, motivated by the instability of the VSM results in this setup and the uncertainty of the overall feasibility of the approach. It would be unwise to optimize the LSA model for computational efficiency when the approach, even with the simpler VSM model, turns out to be impractical.

In Section 4.2.4, we argued that the VSM model seems to work on chains with certain characteristics, even though the Dsim measure does not reflect this. For example, the visual representation of the results of chain 6 in Figure 6.5c indicates that there is a difference between the activated and deactivated corpus, even though Dsim (0,004, -0.009, -0.010) does not express this. Dsim discards too much information, i.e. it does not accurately represent the actual difference between a feature activated and deactivated corpus. Our first intuition on how to use IR techniques for test case selection (see discussion on solution development in Section 2.3) would require to empirically identify a threshold value  $\alpha$  for selecting/discardng a particular test case. Now however, after sampling more feature chains with a realistically sized document corpus, it seems unlikely that we can determine a useful threshold value. Dsim clearly indicates that the difference between a feature activated and deactivated corpus cannot be measured by maximum similarity. Hence, a similarity threshold would not work either.

We therefore decided to reshape the original solution description (Section 2.3), allowing us to define an alternative evaluation metric to Dsim. Instead of using a test case as query and ranking product artifacts, we calculate an aggregated similarity of each test case to a product variant. This

means that each test case is ranked with respect to a particular product variant. We can use this rank, in combination with the test case relevance information provided by a feature chain, to evaluate 1) the relative performance of IR techniques and 2) the test case selection performance. This reformulation of the problem leads to a simplified experimental setup, as illustrated in Figure 6.7. The main differences to the previous two setups (Figure 6.3) are in Step 1 where we create only one feature activated corpus per chain and in Step 4 where all existing system test cases are ranked according to their similarity to the feature activated corpus.

### 4.3 Experimental Setup 3: Ranking

In the previous setups we studied the feasibility and scalability of the approach, leading to refinements in individual steps, to the removal of the LSA model due to its computational cost from the experiment, and to a re-formulation of the problem in order to enable the evaluation. The objectives of this setup are therefore to:

- adapt the experimental setup to the problem reformulation
- choose the correct statistical technique to evaluate and determine the factors that influence the IR technique performance

In this setup, we modified the configuration of the experiment in Steps 1 and 4 (see Figure 6.7 and 6.2). In Step 4, we calculate the similarity of each test case (1600) to a product variant and then use this similarity score to rank the test-cases. Since we know which test cases are relevant for a product variant, we can then evaluate the IR techniques based on the respective test case rankings.

Formally, we have in Experimental Setup 3 ten feature chains  $I_{1\dots 10}$  that include ten features  $F_{1\dots 10}$ . We have ten feature vectors  $A_{a1\dots 10}$  resulting in ten system variants  $V(A_{a1\dots 10})$ . To each feature, one or more test cases are mapped: in total, 65 test cases are mapped to ten features. We rank all 1600 test cases, which include the 65 mapped test cases, according to  $\text{sim}(T_{1\dots 1600}, V(A_{a1\dots 10}))$ . As for independent variables, we manipulate the content of test cases  $T_{1\dots 1600}$  and use two different summary statistics when calculating  $\text{sim}$ .

The consequence of this problem reformulation, compared to Experimental Setup 1 and 2, is an increase in computational cost: the similarity of each test case variant (4,800) to the product variant needs to be calculated (as opposed to the 3 test case variants in Experimental Setup 2). To make the ranking evaluation feasible, we employed two strategies to reduce the size of the document corpus and created two types of corpora:

1. A minimal corpus that contains only the artifacts that are affected by a feature activation, reducing thereby noise stemming from artifacts

that are common to all product variants. This allows us to pilot the new setup with a relatively short run-time.

2. An automatic corpus that represents the configured product as accurately as possible, reducing thereby the number of artifacts to what is actually composing a deployed product.

We call the first corpus “minimal” for two reasons: a) features are implemented in a small subset of the total files, hence the corpus is small compared to the second, “automatic” corpus; b) the difficulty level is minimized by removing noise caused by not relevant files. We call the second corpus “automatic” since we employ techniques to create this corpus without manual intervention, in contrast to the previous experimental setups.

#### 4.3.1 Step 1 – Corpus Creation

For the creation of the minimal corpus, we followed the same procedure as in Experimental Setup 2, i. e. manually tracing the impact of a feature activation to product artifacts. However, in this setup, we only included artifacts that were affected by a feature activation, thereby creating a minimal corpus.

The idea for creating an automatic corpus stems from the disadvantages of manual corpus creation, being inefficient (it must be repeated for every chain), error-prone, and most importantly, incomplete. With a manual configuration, only the traced option is considered and all other options are not implemented in the product artifacts. This means that the artifacts in a manually generated corpus do not correspond to the artifacts that compose a product that would be eventually installed and tested on a camera, leading to a larger, less accurate corpus.

Listing 6.1: Makefile with conditional inclusion of files

---

```
1 [...]  
2 ifeq ($(CONFIG_A), y)  
3 ide-gd_mod-y += ide-disk.o ide-disk_ioctl.o  
4 endif  
5 [...]  
6 ifeq ($(CONFIG_B), y)  
7 ide-gd_mod-y += ide-floppy.o ide-floppy_ioctl.o  
8 endif  
9 [...]
```

---

Listing 6.2: Preprocessor directives with conditional inclusion of files and code

---

```
1 [...]  
2 #ifdef CONFIG_D  
3 #include <asm/irq.h>
```

---

```

4 #endif
5 [...]
6 void led_classdev_unregister(struct led_classdev *led_cdev)
7 {
8 #ifdef CONFIG_C
9 if (led_cdev->trigger)
10 led_trigger_set(led_cdev, NULL);
11 #endif
12 [...]
13 }

```

---

For example, Listing 6.1 shows an excerpt from a Makefile where a configuration option (line 2) determines whether 2 files (line 3) should be built or not. If CONFIG\_A is the traced option, a manual configuration would delete the source files corresponding to line 2. However, independently of whether CONFIG\_B in line 6 is activated, the corresponding files in line 7 would be included in a manually generated corpus.

The same principle holds for preprocessor directives that realize configuration options. In Listing 6.2, assume CONFIG\_C to be the traced option (line 8). With a manually generated corpus, the configuration option in line 2 would not be evaluated, therefore including the file (line 3) into the corpus, independently whether the product is actually run on an ARM processor. This increases the size of the corpus, adds noise and does not reflect the product for which the system test cases were developed.

We addressed this issue by exploiting the product build system. The basic idea of our approach is to hook into the build process custom code that performs pre-processing operations on the files included in the product. As a result, we get the configured (preprocessed) source code and the compiled product as it is installed on the camera, including configuration files and documentation. In this way, we could create a corpus that corresponds to the tested product merely by configuring and building the product.

Table 6.5 illustrates the size characteristics of the minimal and automatic corpus from this setup and the manual corpus from Experimental Setup 2. We reduced the average size for the automatic corpus by a factor 6 compared to the manual corpus. The minimal corpus is significantly smaller, which allowed us to pilot the ranking approach.

The second major difference to Experimental Setup 2 was to use all system test cases (1,600) and rank them instead of analyzing similarity between a product and the applicable test case(s). As test cases were stored in a database, we could easily automatize the construction of the 4,800 test case variants.

#### 4.3.2 Step 2 – Term Extraction

We applied the same process as in Experimental Setup 2.

Table 6.5: Corpora sizes (# of documents / # of terms)

| <i>Chain</i> | <i>Minimal corpus</i> | <i>Automatic corpus</i> | <i>Manual corpus (setup 2)</i> |
|--------------|-----------------------|-------------------------|--------------------------------|
| 1            | 10/1,668              | 11,078/68,619           | 67,332/347,854                 |
| 2            | 3/260                 | 11,078/68,613           | 67,331/341,098                 |
| 3            | 56/3,471              | 11,078/68,614           | 67,332/347,853                 |
| 4            | 2/150                 | 11,078/68,613           | 67,334/359,954                 |
| 5            | 2/29                  | 11,078/68,613           | 67,331/341,097                 |
| 6            | 29/1,638              | 11,078/68,611           | 67,332/347,857                 |
| 7            | 61/2,170              | 11,078/68,608           | N/A                            |
| 8            | 2/242                 | 11,078/68,611           | N/A                            |
| 9            | 3/630                 | 11,078/68,613           | N/A                            |
| 10           | 3/66                  | 11,078/68,612           | N/A                            |

#### 4.3.3 Step 3 – Similarity Analysis

In Experimental Setup 2, a similarity analysis for one product configuration and one test case variant required approximately one hour of computational time. With 4,800 test case variants, this would amount to a computation time of 200 days per chain. However, with the reduced corpus sizes we could compute a chain within 2 (minimal corpus) respectively 17 (automatic corpus) hours, making the ranking approach feasible.

#### 4.3.4 Step 4 – Evaluation

The evaluation procedure differs considerably from Experimental Setup 2 where we looked at the difference of a feature activated and deactivated corpus. In this setup, we have only a feature activated corpus and rank the test cases according to their similarity to that corpus. To create such a test case ranking, we compute a single similarity value using a summary statistic of the corpus' documents similarity to each test case. As summary statistics we chose maximum and mean similarity.

The independent variables in this setup are the test-case variants and the summary statistics. For the test case variants we have three levels (see Table 6.2) while the summary statistics consist of two levels (maximum and mean). Since we have two independent variables and use the same data for all factors, we have a two-way repeated measures design. The dependent variable is the ranking position of a test-case, i. e. the lower the ranking position the better. In total, we have 65 ground truth instances, i. e. relevant test cases that are ranked between position 1 and 1600.

The evaluation goal in this setup is twofold: (1) we compare the effect of the independent variables on the ranking performance; (2) we study the ranking performance to understand whether it can support test case selection decisions.

(1) COMPARISON OF RANKING PERFORMANCE We formulate the following null hypotheses:

$H_0_1$ : The test case content does not significantly affect the ranking performance of the VSM.

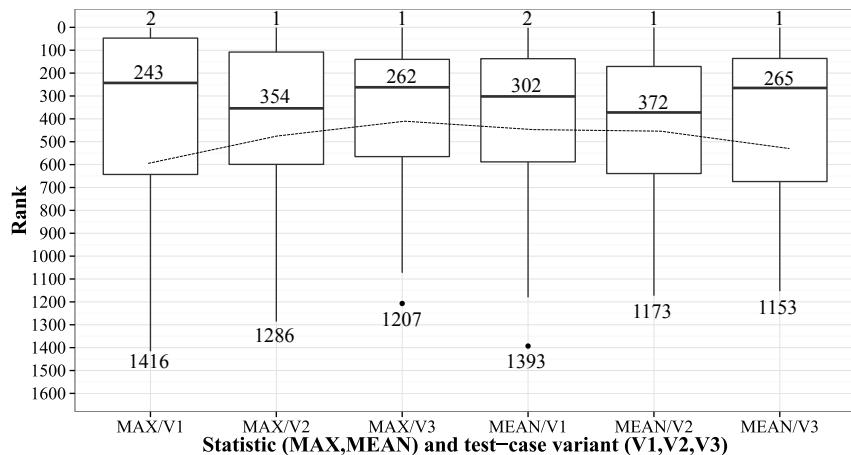
$H_0_2$ : The summary statistic does not significantly affect the ranking performance of the VSM.

Furthermore, we evaluate whether the size of the corpus impacts the ranking performance and formulate the third null hypothesis as follows:

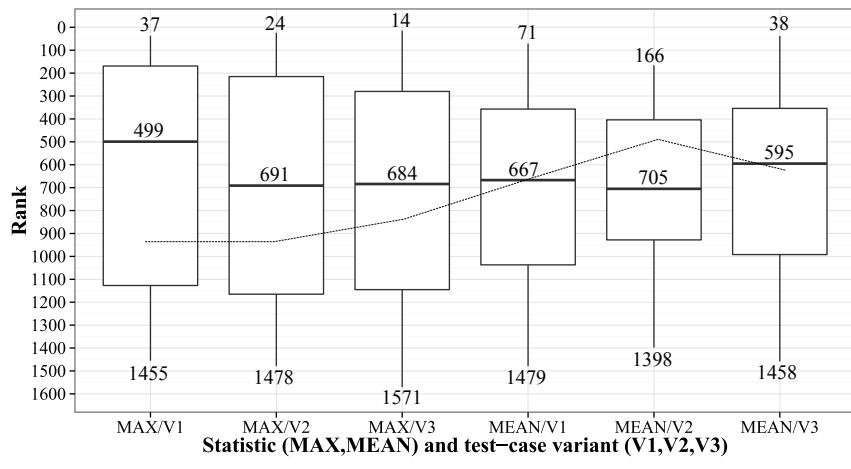
$H_0_3$ : The corpus type does not significantly affect the ranking performance of the VSM.

Figure 6.8 illustrates the ranking results in the form of box plots. The y-axis shows the ranking position (the numbers indicate minimum, median and maximum position) of the 65 test cases whereas the different box plots represent the two factors with two and three levels. Even though in both corpus types a large variance can be observed, the spread is more pronounced in the automatic corpus. This is an expected result as the minimal corpus contains less noise than the automatic corpus. Looking at the minimal corpus in Figure 6.8a, one can observe that, for the maximum statistic, the more specific the information in the test case the smaller is the variation (inter-quartile range, dashed line) in ranking results. This behavior is reversed for the mean statistic where the inter-quartile range increases with the specificity of the test case content. Looking at the automatic corpus in Figure 6.8b, the impact of statistic and test case content on inter-quartile range is *not* analogous to the one observed in the minimal corpus. This is an indication that the characteristics of the corpus (minimal vs. automatic) affect IR technique performance, not only by overall absolute values, but also in terms of the impact of the summary statistic and the test-case input on the ranking.

In order to test the stated hypotheses we used randomization tests [Ludbrook and Dudley, 1998, Edgington and Onghena, 2007] provided by the ez package [Lawrence, 2013] for R. Table 6.6 lists the results of testing  $H_0_1$  and  $H_0_2$ , which we consequently fail to reject at  $\alpha = 0.05$ : neither test case content nor summary statistic significantly affect the test case ranking performance, both on the minimal and automatic corpus. However, we reject  $H_0_3$  with a p-value  $< 0.001$  (randomization test, 100,000 iterations).



(a) Minimal corpus



(b) Automatic corpus

Figure 6.8: Box plots of test case rankings – the dashed line indicates the interquartile range differences

Table 6.6: Results of randomization tests (100,000 iterations) of hypotheses  $H_0_1$  and  $H_0_2$

| <i>Effect</i>        | <i>p-value minimal corpus</i> | <i>p-value automatic corpus</i> |
|----------------------|-------------------------------|---------------------------------|
| 1. Test case content | 0.32022                       | 0.11216                         |
| 2. Summary statistic | 0.07580                       | 0.98244                         |

## (2) TEST CASE SELECTION DECISION SUPPORT

Table 6.7: Test case ranking positions and Mean Average Precision

| <i>Chain</i> <sup>1</sup>                     | MAX/V <sub>1</sub> | MAX/V <sub>2</sub> | MAX/V <sub>3</sub> | MEAN/V <sub>1</sub> | MEAN/V <sub>2</sub> | MEAN/V <sub>3</sub> |
|---|--------------------|--------------------|--------------------|---------------------|---------------------|---------------------|
| Minimal corpus – test case ranking position   |                    |                    |                    |                     |                     |                     |
| <b>1 (1)</b>                                  | 57                 | 142                | 311                | 81                  | 327                 | 136                 |
| 2 (2)   | 1,416              | 1,286              | 955                | 1,393               | 1,166               | 992                 |
| <b>3 (5)</b>                                  | 111                | 85                 | 171                | 41                  | 16                  | 136                 |
| <b>4 (1)</b>                                  | 2                  | 1                  | 4                  | 2                   | 1                   | 4                   |
| <b>5 (1)</b>                                  | 656                | 481                | 319                | 656                 | 481                 | 319                 |
| <b>6 (1)</b>                                  | 37                 | 107                | 105                | 4                   | 28                  | 125                 |
| <b>7 (34)</b>                                 | 4                  | 11                 | 20                 | 92                  | 114                 | 58                  |
| 8 (10)  | 891                | 766                | 660                | 912                 | 795                 | 670                 |
| <b>9 (7)</b>                                  | 221                | 357                | 171                | 168                 | 232                 | 181                 |
| <b>10 (3)</b>                                 | 29                 | 9                  | 17                 | 20                  | 5                   | 11                  |
| MAP   | 0.1140             | 0.1913             | 0.0879             | 0.1287              | 0.2032              | 0.0949              |
| Automatic corpus – test case ranking position |                    |                    |                    |                     |                     |                     |
| <b>1 (1)</b>                                  | 461                | 524                | 165                | 497                 | 630                 | 1,365               |
| 2 (2)   | 174                | 203                | 226                | 301                 | 346                 | 509                 |
| <b>3 (5)</b>                                  | 67                 | 185                | 14                 | 229                 | 194                 | 631                 |
| <b>4 (1)</b>                                  | 46                 | 42                 | 117                | 820                 | 736                 | 376                 |
| <b>5 (1)</b>                                  | 888                | 805                | 1,112              | 1,074               | 793                 | 821                 |
| <b>6 (1)</b>                                  | 290                | 359                | 537                | 357                 | 182                 | 635                 |
| <b>7 (34)</b>                                 | 892                | 1,120              | 1,229              | 71                  | 225                 | 38                  |
| 8 (10)  | 77                 | 45                 | 123                | 236                 | 392                 | 200                 |
| <b>9 (7)</b>                                  | 37                 | 66                 | 109                | 477                 | 450                 | 347                 |
| <b>10 (3)</b>                                 | 117                | 138                | 172                | 876                 | 656                 | 354                 |
| MAP   | 0.0116             | 0.0122             | 0.0086             | 0.0058              | 0.0057              | 0.0062              |

<sup>1</sup> The number in parentheses indicates the total number of test cases that were mapped to that particular chain.

Table 6.7 shows a subset of the results (for each chain only the test case with the highest rank is shown) in this experimental setup, both for the minimal and the automatic corpus. The first column indicates the chain and the number of test cases identified by the test engineers as being relevant for that feature. The remaining columns refer to the rank (out of 1,600) with the given test case variant and used summary statistic. For example, in

Table 6.8: Quality of the ranking results, judged by test area maintainers

| <i>Result / Chain</i>    | <b>1</b> | <b>3</b> | <b>4</b> | <b>6</b> | <b>7</b> | <b>10</b> | <i>Aggregate</i> |
|--------------------------|----------|----------|----------|----------|----------|-----------|------------------|
| <i>Total test cases</i>  | 48       | 29       | 25       | 19       | 141      | 33        | 295              |
| <i>True positives</i>    | 9        | 12       | 7        | 4        | 48       | 21        | 101              |
| <i>False positives</i>   | 39       | 17       | 18       | 15       | 93       | 12        | 194              |
| <i>Average Precision</i> | 0.21     | 0.61     | 0.42     | 0.24     | 0.10     | 0.70      | <b>0.38</b>      |

the minimal corpus in Table 6.7, the only test case in Chain 1 was ranked on position 81, using the complete test case and the mean as summary statistic. A lower rank indicates a better result since a relevant test case would appear higher in the ranked list of test cases.

Table 6.7 shows also the mean average precision (MAP) [Liu, 2011] of the test case rankings. MAP allows us to interpret how useful the ranking is for the task at hand. MAP is calculated by taking the mean of the average precision (AP) over a set of queries. AP is calculated by averaging the precision at each relevant test case. For example, representing correct results as 1 and 0 otherwise, the AP for the query result 0 1 1 0 0 0 1 is  $(1/2 + 2/3 + 3/7)/3 = 0.53$ . The MAP is then calculated by averaging the AP of a series of query results. A MAP of 0.1 means that only every tenth ranked item is relevant, whereas with a MAP of 0.5, every second item is relevant.

In order to better understand the quality of the achieved rankings, we involved test engineers in the evaluation. Unfortunately, no test history was available for the studied product and feature chains, not allowing us to compare the achieved ranking with the selection from test engineers. However, by querying test engineers we could establish whether test cases associated with a particular feature are relevant or not. For example, in Chain 1, with the maximum summary statistic and test case variant 1, the known to be relevant test case was ranked on position 57 (see Table 6.7, minimal corpus). This means that 56 test cases were ranked above the relevant one. Our question to the test engineers was whether these higher ranked test cases were indeed relevant for the particular feature. We chose those chains for evaluation where the relevant test case was ranked below position 100. These six chains are indicated in bold typeface in Table 6.7. Then, for each relevant test case (65), we selected those test cases which were ranked higher, or, in case the relevant test case was ranked at position 20 or lower, all top 20 test cases. This resulted in a set of 295 test cases, for each of which we identified the corresponding test area maintainers (eight in total). Table 6.8 shows the true and false positive rates for each chain. According to the MAP in Table 6.7, the IR technique with the mean

summary statistic and variant 2 of the test case performed best. Hence, we chose that test case ranking to calculate the average precision, shown in Table 6.8.

#### 4.3.5 Discussion and Lessons Learned

We start this section with a discussion of the results from Experimental Setup 3, addressing RQ-2 and RQ-3, and then elaborate on the lessons learned from this evaluation, addressing RQ-1.

The first goal of this experimental setup was to study whether the test case content or summary statistic has an effect on the ranking performance of the VSM. The results (Table 6.6) suggest that there is no statistically significant evidence for such an effect, even though there is a strong significant evidence that the ranking performance on minimal and automatic corpus differ (see the rejected H<sub>0</sub><sub>3</sub>). Looking at the results summarized in Table 6.6, one can observe in the minimal corpus a tendency that the summary statistic affects the ranking performance. In the automatic corpus this tendency disappears. Even though this result is not statistically significant, it illustrates an important point: the characteristics of the data set on which the IR techniques are evaluated matter, even to the extent where the impact of a factor is reversed. Neglecting this can lead to sub-optimal IR technique configurations, e.g. when they are evaluated on a data set with characteristics that differ from the one where the technique is eventually applied for productive use. The experiments conducted by Falessi et al. [2013] illustrate the effect of data set difficulty on the performance of different IR techniques, confirming our observation that under different corpus characteristics (minimal vs. automatic), IR techniques behave differently.

The second goal of this experimental setup was to study to what extent the ranking provides test case selection decision support. Looking at Table 6.7, the minimal corpus, we can observe that for Chains 1, 3, 4, 6, 7 and 10 ranking positions below 100 were achieved. According to the MAP measure, the combination of V2 (test case except test steps) and the mean summary statistic would provide the best performance (i.e. every 5th test case is relevant). However, looking at the automatic corpus, the combination of V2 and the maximum summary statistic would provide the best performance. Furthermore, the MAP measure in the automatic corpus is by a factor 10 lower than in the minimal corpus, supporting the above conclusion that data set difficulty affects IR technique performance. We also evaluated how useful the best achieved rankings would be for test case selection. Table 6.8 illustrates the achieved average precision with the selected best IR technique configuration. The results suggest that for some feature chains the approach works very well, e.g. for Chain 3 and 10 with a MAP > 0.5, which means that every second test case inspected by the test engineer was relevant. On the other hand, in Chain 7 only every 10th

test case was assessed as relevant. Furthermore, we note that Table 6.8 shows only the six out of ten feature chains for which we could achieve a ranking  $< 100$  of one or more known to be relevant test cases. While these result seem underwhelming, they are comparable w.r.t. the achieved MAP of 0.38 to similar studies aiming at automatic traceability recovery. Abadi et al. [2008] traced code to documentation and experimented with 17 IR technique configurations, achieving a MAP between 0.04 and 0.66. Cleary et al. [2009] compared 6 concept location techniques, achieving a MAP between 0.006 and 0.06 (they attribute the low values to the large data set). Qusef et al. [2014] combine IR techniques with dynamic slicing to identify the classes tested by a test suite, achieving a MAP between 0.83 and 0.93. More recently, Xia et al. [2015] associated bug reports with developers, achieving a MAP of 0.51.

We turn now to the lessons learned from the statistical analysis. When we perform experiments comparing different IR techniques, we exercise them on the same set of tasks and determine the techniques' relative performance by testing whether their location parameter (e.g. mean or median) of some performance measure differs significantly [Smucker et al., 2007]. This general IR technique evaluation framework guides the selection and configuration of permissible statistical analyses, leading to the following considerations:

1. *Test assumptions*: The characteristics of the population distribution from which the dependent variables are sampled determine whether the assumptions (normality, independent samples, homogeneity of variance) of parametric tests are violated. An assumption that is likely to be violated is normality, caused by a skew to the right of the dependent variable. This is due to the inherent properties of some IR performance measures, e.g. a ranking cannot be negative.
2. *Repeated measures*: When comparing IR techniques we apply them on the same corpus, looking at the effect of different treatments (IR techniques) on the same subject (corpus). In other words, we have a paired (two IR techniques) or repeated (more than two IR techniques) measures design.
3. *Multiple testing*: When many different configurations of IR techniques are compared, we encounter the multiple testing problem [Bender and Lange, 2001]: the more statistical tests are performed, the higher the likelihood that a true null hypothesis is rejected (false positive or Type I error).

Individually, the above considerations can be addressed by choosing statistical procedures fulfilling the given requirements. For example, if the assumptions of the parametric tests are violated, one can use non-parametric or distribution-free alternatives [Sheskin, 2000]. For a repeated measures

design, paired difference tests or repeated measures ANOVA are viable choices [Sheskin, 2000]. The multiple testing problem can be addressed by applying adjustments to p-values or, depending on the particular study design, choosing test procedures that compensate for multiple comparisons [Bender and Lange, 2001].

In the context of evaluating IR techniques, we need to address all of the above stated considerations simultaneously and select a statistical procedure whose assumptions are not violated, allows for a repeated measures design and provides means to compensate for multiple testing. It turns out that these combined requirements are difficult to fulfill. If we assume that the normality assumption is violated, requiring a non-parametric test, and that we have a repeated measures design, we can only evaluate designs with at most one factor using Friedman's two-way analysis of variance by ranks [Sheskin, 2000]. This means that factorial designs to compare IR techniques can not be effectively evaluated with traditional statistical means, without allowing for violations of the test procedure and accepting a potential loss of statistical power.

For example, Biggers et al. [2014] ignore potential normality violations (the dependent variable is a rank based on a similarity measure) and use regular ANOVA with five factors<sup>8</sup>. Subsequent analyses of interactions and main effects are performed with Kruskal-Wallis analysis of variance by ranks [Sheskin, 2000] (a non-parametric test) which indicates that the authors were aware of the potential violations of the parametric test assumptions. Further examples from recent journal publications illustrate the inherent difficulty to correctly evaluate IR techniques:

- Poshyvanyk et al. [2012] use Wilcoxon matched-pairs signed-ranks test [Sheskin, 2000], acknowledging a potential violation of normality of their performance measure and taking advantage of the stronger statistical power of the paired test; however they do not compensate for multiple testing when evaluating the effect of stemming on the performance measure (48 tests).
- Thomas et al. [2013] use Tukey's HSD test which is commonly recommended when all pair-wise comparisons in a data set need to be evaluated [Sheskin, 2000]. This test compensates for multiple tests, assumes however normality and homogeneity of variance. Even though the authors tested for the latter, normality is silently assumed.
- Falessi et al. [2013] provide guidelines on the statistical evaluation of IR techniques, motivating the use of inferential statistics [Sheskin, 2000] by the need to "check whether the observed difference could reasonable occur just by chance due to the random sample used for

---

<sup>8</sup> Unfortunately it is not possible to determine whether this potential assumption violation had an impact on the outcome of the analysis since the raw data for this particular part (Part 1) of the case study has not been published by the authors.

developing the model” [Falessi et al., 2013]. However, in the case-study presented in the same paper [Falessi et al., 2013] they refrain from using inferential statistical tests and rely on descriptive statistics to determine the best IR technique, not following their own best practice principles<sup>9</sup>.

We chose to follow the advise by Smucker et al. [2007] and used the randomization test [Ludbrook and Dudley, 1998, Edgington and Onghena, 2007, Mittas and Angelis, 2008] to perform our IR technique comparison. The major advantage of randomization tests is that they do not assume any particular theoretical population distribution from which the sample is drawn but rather construct a sample distribution by permutations of the observed data [Sheskin, 2000]. This means that the skewness of our dependent variable (rank) is of no concern. A disadvantage of randomization tests is their computational cost: we performed the procedure with 100.000 permutations on 65 observations on two factors ( $2 \times 3$  levels) in 11 hours. This is a considerable cost compared to generally instantaneous results of traditional statistical tests.

Randomization tests are also applicable to repeated measure designs [Sheskin, 2000]. However, procedures to address the multiple testing problem in the context of repeated measure designs are difficult to implement, since comparisons occur between-subject factors, within-subject factors, or both [Bender and Lange, 2001]. Hence, to the best of our knowledge, multiple testing is still an open issue, in the context of IR technique comparisons with more than two factors and under the constraints of a repeated measures design and potentially violated normality assumptions.

## 5 CONCLUSIONS AND FUTURE WORK

This chapter illustrated the experiences and lessons learned from developing and evaluating an approach for test case selection decision support based on Information Retrieval (IR) techniques. We rooted the solution development in the context of a large-scale industry experiment which allowed us to evaluate IR techniques in a realistic environment. The solution development was guided by incremental refinements of the experimental setup, both testing the scalability and performance of IR techniques. In order to provide insight for researchers, we reported on the design decisions, both in terms of experimental setup and IR technique implementation. We conclude by providing answers to the initially stated research questions.

---

<sup>9</sup> They use inferential statistics to compare the best IR technique with the optimal combination of IR techniques, but this comparison is questionable since the preceding selection of best technique is based on descriptive statistics.

**RQ-1 TO WHAT EXTENT CAN STATE-OF-THE-ART IR TECHNIQUES BE APPLIED IN A LARGE-SCALE INDUSTRIAL CONTEXT?** We identified a set of open questions that need to be addressed in order to bridge the gap between laboratory IR experiments and applications of IR in industry:

- Comparative studies on IR techniques seldom optimize the parameters of all underlying IR models, leading to potentially unfair evaluations. This can even go as far as to reporting contradictory results, as we have shown in Section 4.1.5, and has been also observed by Grant et al. [2013]. This poses a threat to both researchers and practitioners who might adopt IR techniques based on flawed evaluations. Future work is required on evaluating IR techniques both on standardized data sets but also on large-scale, industry-grade data.
- Parameter optimization is computationally expensive, particularly when applied on industry-grade, large data sets. Possible avenues for enabling parameter optimization efforts are: (1) investigating means to reduce the number of terms in a corpus, without affecting the IR models performance. Such techniques would however be very application and data specific, e. g. as the fact extraction and corpus-size reduction techniques illustrated in Sections 4.2.1 and 4.3.1 respectively; (2) using parallel/distributed implementations of the algorithms, e. g. Singular Value Decomposition, that power IR techniques. A third alternative would be to compare optimized parametric with non-parametric IR models, such as Hierarchical Dirichlet Processes [Teh et al., 2006]. If non-parametric models would show to be consistently equivalent or better than parametric models, they would be the favorable solution in an industrial environment since they would require less customization effort.
- Determining the superiority of one IR technique over another should be based upon inferential statistical techniques [Falessi et al., 2013]. However, as we have elaborated in Section 4.3.5, developing and evaluating a valid statistical technique that allows simultaneously for non-normal dependent variables, a repeated measures design and multiple comparisons, in the context of IR evaluation, is still an open issue.

**RQ-2 TO WHAT EXTENT DO THE USED SOFTWARE DEVELOPMENT ARTIFACTS INFLUENCE THE PERFORMANCE OF IR TECHNIQUES?** We have experimented with different variants of both test case content, i. e. information within the test case description, and product artifacts, in particular the size of the document corpus. We could not determine a statistical significant performance difference between test case variants. However, we observed that the size of the corpus influences the performance considerably (observed in Experimental Setup 2, and shown with statistical signif-

icance in Experimental Setup 3). This confirms the observations by [Falessi et al. \[2013\]](#) who evaluated IR techniques at varying difficulty levels. As a consequence, this means that IR techniques need to be evaluated on realistic, large dataset that mirror the difficulty of datasets encountered in real industry applications.

**RQ-3 TO WHAT EXTENT CAN THE STUDIED IR TECHNIQUES SUPPORT TEST CASE SELECTION?** We have evaluated the test selection performance on ten feature chains. In four chains, the feature “signal” in the product artifacts was too weak to induce an actionable ranking. In the other six feature chains, we tasked test engineers to rate the test case ranking with respect to their relevance for the corresponding feature. The results in terms of precision ( $MAP = 0.38$ ) are comparable to what has been achieved in similar studies that aimed at automating trace recovery (e.g. [Abadi et al. \[2008\]](#), [Cleary et al. \[2009\]](#), [Qusef et al. \[2014\]](#), [Xia et al. \[2015\]](#)). However, while our results reflect the state of art, the approach based on IR techniques is not yet reliable enough to automate the test case selection process in a realistic industry setting.

In future work, we plan to study the properties of the features chains where the differentiation between a feature activated and deactivated corpus failed. Knowing these properties and being able to detect them in the corpus would allow us to provide suggestions on how to improve source code documentation, e.g. by better naming conventions using a company-wide glossary. Furthermore, by including meta-data from the version control system in the corpus, such as commit comments connected to feature activation/deactivation code, we could improve the test case rankings.

# 7

---

## SUPPORTING EXPERTS IN TEST CASE SELECTION WITH TOPIC MODELS

---

**SUMMARY** Test case selection is one possibility to focus the regression testing effort to the part of the system under test which is likely to exhibit faults. While there exist many test case selection techniques, the majority relies upon structural knowledge of the system or detailed design documentation. Our goal is to support test engineers in their selection task by combining their domain expertise with the ability to analyze large amounts of textual data originating from system test cases. We create topic models and exploit their probabilistic nature to reorganize an existing test case categorization, facilitating the discovery of relevant but unexpected test cases from a pool of existing test cases. We developed RiTTM, risk-based testing supported by topic models, in collaboration with our industry partner where we also evaluated the approach in a case study. RiTTM was applied by a non-domain expert on a database containing 1415 test cases, and identifying 29 relevant test cases that were not chosen originally by a domain expert test engineer. Seventeen of these test cases were serendipitous, that is, they were found to be unexpected.

### 1 INTRODUCTION

Regression testing has the purpose of verifying that changes to existing software did not introduce faults that affect the behavior of the unchanged parts of the software [Yoo and Harman, 2012]. Often, time and resource constraints prohibit the re-execution of all test cases to quality assure the software, requiring that only a subset is selected [Rothermel and Harrold, 1996]. This problem is even more pronounced in system testing [Petschenik, 1985], whose purpose is to quality assure the software before it is delivered to the customer [Whittaker, 2000]. To achieve competitive time-to-market cycles, it is necessary to perform system test case selection that balances fault slip-through [Damm et al., 2006] to the customer and testing cost.

The majority of test selection techniques are modification-aware: test cases are selected because they are relevant to the modified part, requiring structural knowledge, e.g. source code or call graphs, of the system under test [Yoo and Harman, 2012]. Another set of test selection methods is based on historical test outcomes but this limited information means they can only exploit general trends and recent failure patterns [Feldt, 2014]. A few techniques are design based [Yoo and Harman, 2012] and rely on the availability of specific artifacts or models (e.g. Chen et al. [2002], Muccini et al. [2006], Briand et al. [2009], Wang et al. [2013]). However, a recent survey among practitioners indicates that models, if at all used, serve rather as a communication aid and are therefore seldom updated and maintained over time [Gorschek et al., 2014].

In previous work [Unterkalmsteiner et al., 2015] (see Chapter 6), we traced source code to system test cases and experimented with test case selection based on changes in source code. The mixed results, both in terms of selection and runtime performance, led us to investigate alternatives that were viable at our industry partner and do not rely upon a formal documentation of product variants. In this chapter, we present risk-based testing supported by topic models (RiTTM), a system test case selection approach that provides decision support based on existing artifacts and resources that are readily available. RiTTM does not introduce new documentation or models that require manual maintenance. Furthermore, the approach integrates the test engineer in the test selection decision process, counteracting the out-of-the-loop phenomenon [Parasuraman et al., 2000]. We have evaluated RiTTM in a case study, aimed at determining the feasibility of the approach, but also to study its ability to identify new but relevant test cases. Indeed, our primary motivation for developing RiTTM was to study the potential of topic models [Blei et al., 2003, Steyvers and Griffiths, 2007, Blei, 2012] to facilitate discovery that is synthesized into insight [André et al., 2009]. While this idea has been studied recently in the context of recommender systems (e.g. Zhang et al. [2012], Garcin and Faltings [2013], Taramigkou et al. [2013], Alexander et al. [2014]), it is, to the best of our knowledge, novel in the field of software engineering. In addition to RiTTM itself, this chapter makes the following contributions:

- We illustrate the application of a practical and automated parameter tuning process to perform topic model selection.
- We evaluate intrinsic model quality (for model selection) using a metric that is adequate for the task for which the model is eventually used. We evaluate the extrinsic quality of the model in an industry case study.
- In the case study, we show that RiTTM can be integrated into a industry scale regression test process.

The remainder of this chapter is structured as follows. Section 2 reviews background and related work. In Section 3 we introduce RiTTM and motivate its overall design. We illustrate the case study design in Section 4 while the results are presented and discussed in Section 5. The chapter concludes in Section 6, pointing out directions for future work.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Risk-based Testing

The notion of risk is an integral part of the test planning process [ISO, 2013]. Risk-based testing is a pragmatic approach, widely applied in industry, to address software complexity which goes hand in hand with an increased testing effort [Felderer and Schieferdecker, 2014]. The basic premise of risk-based testing is to focus testing on functions with the largest risk exposure [Amland, 2000]. Risk exposure, in general, is determined by the probability of a negative outcome and the impact of such an event [Boehm, 1991]. In the context of risk-based testing, this translates to the probability of the occurrence of a fault and its cost if it occurs in production [Amland, 2000]. The fault probability can be gauged by the nature of the functionality under consideration, e. g. number and amount of changes, history in fault proneness, experience of the contributing developers, or the maturity of the development process [Felderer and Schieferdecker, 2014]. The cost of a fault in production can be estimated by the value the functionality under consideration provides to the user of the product. The estimation for these risk exposure factors can be based on a formal model (e. g. Li et al. [2006]) or based on subjective expert opinion (e. g. Felderer and Ramler [2014]).

It is important to point out that risk-based testing does not aim at reducing the amount of testing. While test effort reduction can be a consequence of techniques that predict fault proneness of a product [Elberzhager et al., 2012], the goal of risk-based testing is to increase the probability of identifying faults in functionality that is crucial for customer satisfaction by focusing the test effort *and* reducing risk exposure [Felderer and Ramler, 2014]. The means by which risk-based testing can be implemented vary considerably. Felderer and Schieferdecker [2014] provide a taxonomy of risk-based testing that is useful for the discussion and analysis of different approaches. We focus on the risk-based test planning process, in particular on test case prioritization and selection, since this is the process we aim to support in our case company (see Section 4.1).

## 2.2 Test Case Selection

Test case selection is a central activity both in the planning process of risk-based testing [Felderer and Schieferdecker, 2014] and in regression testing [Yoo and Harman, 2012]. In both scenarios, the goal is to choose a subset of test cases because the execution of the whole set is impractical due to time or resource constraints. The problem of selecting a subset of test cases from a larger set is similar to test suite minimization. Test case selection focuses however on changes in the system under test rather than optimizing functional coverage with a minimum amount of test cases [Yoo and Harman, 2012].

There exists an abundance of techniques for test case selection, surveyed by Yoo and Harman [2012] and Engström et al. [2010]. One criterion on which these techniques can be classified is whether they use implementation information of the system under test or not. White box test case selection techniques exploit, for example, changes in source code, analyze data and control flows, and execution traces [Yoo and Harman, 2012]. Black box test case selection techniques are less common [Engström et al., 2010] and use, for example, changes in design documentation to perform impact analysis [Yoo and Harman, 2012]. Since our proposed approach to system test case selection does not rely upon information from the system under test, we consider it a black-box approach. As related work, we discuss therefore other black box approaches.

Chen et al. [2002] use activity diagrams to represent the desired system behavior (requirements), tracing them to test cases. Upon changes in the requirements, they identify the affected entities in the activity diagram and select the corresponding test cases for execution. They complement their approach with risk-based test case selection. Their risk model considers the cost of a fault seen by the customer and the probability that a test case discovers a fault, resulting in a risk exposure score that can be used to prioritize test cases.

Xu et al. [2005] propose to use a fuzzy expert system instead of source code to select system test cases, motivated by security reasons but also for independent verification and validation. Interestingly, their causal model to construct the expert system includes factors that are related to risk exposure, namely defect impact and coverage of changed functions. Other factors considered in the model are schedule and test setup constraints. The results of their case study indicates that the test selection by expert system detects bugs earlier than the original plan.

Sherriff et al. [2007] exploit change records to identify associated files, using this information to select test cases that are traced to historic field failures. The advantage to white box approaches is that their method is independent of the particular programming languages or technologies since it considers only meta-data (change history) of files. On the other hand,

it requires traceability between test cases and the faults discovered by the test cases. A simpler variant is to keep a cache of source code files changed and prioritize and select test cases based on what is in the cache. Practical evaluation on industrial data showed the very simple method to have attractive properties for industry [Wikstrand et al., 2009].

History-based test case prioritization exploits patterns in previous test outcomes to help predict future failures and can thus select only the test cases that are likely to fail [Kim and Porter, 2002]. A benefit is that the information needed is independent of language and actual technologies used; as long as there is a log of previous testing results general patterns and trends can be analyzed statistically. This makes these set of techniques very general. Recently, it was shown that the age of system test cases for a large, real-world software system could help predict the failure probabilities and thus provide important background information to more detailed test selection processes [Feldt, 2014].

In the context of product line engineering, Wang et al. [2013] propose to use feature models to guide test case selection. They argue that a feature model can capture product expertise, and by connecting the feature to a test model, increase test selection efficiency and coverage. However, one disadvantage of this approach is that the feature and test model need to be created if they do not exist. Unfortunately the authors do not quantify this required upfront effort.

### 2.3 Topic Models

Probabilistic topic models are a suite of algorithms that automatically annotate documents with themes, creating a new structure through which the document collection can be explored [Blei, 2012]. It is important to point out that these themes or topics are not created by a human annotator but emerge from creating a probabilistic model of the document collection. The basic idea of topic models is that “documents are mixtures of topics, where a topic is a probability distribution over words” [Steyvers and Griffiths, 2007]. Topic models have been applied to a variety of tasks, e.g. image labeling [Fei-Fei and Perona, 2005], information retrieval [Wei and Croft, 2006], multi-document summarization [Haghghi and Vanderwende, 2009], word sense discrimination [Brody and Lapata, 2009] and pattern recognition [Niu et al., 2012].

The root of probabilistic topic models can be found in another information retrieval algorithm based on dimensionality reduction, Latent Semantic Indexing (LSI) [Deerwester et al., 1990], whose performance improvements with respect to the Vector Space Model (VSM) [Salton et al., 1975] were studied from a probabilistic perspective by Papadimitriou et al. [1998]. Hofmann [1999] expanded LSI to a probabilistic model, pLSI, exper-



Figure 7.1: Topic model example

imentially showing that the probabilistic approach outperforms LSI which is based on linear algebra. pLSI has however two major drawbacks: the number of model parameter grows linearly with the size of the document collection causing over-fitting, and the model does not explain how to assign probability to unseen documents [Blei et al., 2003]. Therefore, Blei et al. [2003] proposed Latent Dirichlet Allocation (LDA) as a probabilistic topic model to overcome these fundamental restrictions of pLSI. LDA has, since its inception, seen a number of extensions and specializations, e. g. hierarchical topic models [Griffiths et al., 2003] and correlated topic models [Blei and Lafferty, 2005] and dynamic topic models [Blei and Lafferty, 2006].

LDA is a mixed-membership model in which each document can be associated with multiple topics in different proportions [Blei, 2012]. To provide an intuition of this concept, we illustrate in Figure 7.1 an example from our case study where we applied LDA to a set of system test cases. The actual test case content and the topic terms have been obfuscated for confidentiality reasons. Note however that this does not detract from the

analysis we are interested in. The figure shows four topics and their distribution over three system test cases. Each topic is defined by a set of terms, ordered by their probability under the topic. We selected these test cases for exemplification as they exhibit interesting patterns in terms of topic distribution<sup>1</sup>, relevant for test case selection support:

- Test case #293 has a dominant topic (Topic 6 with 66%) and a secondary topic (Topic 11 with 16%).
- Test case #335 has a dominant topic (Topic 11) and two secondary topics (Topic 46 and 92).
- Test case #87 has one dominant topic (Topic 46).

We can exploit this information to support test case selection. For example, one could choose a representative test case from each topic to cover as much functionality as possible. Another strategy could be to focus testing only on one particular topic. For example, if a tester wants to quality assure Topic 46 in Figure 7.1, he would select test case #87, but maybe also test case #335 since it seems to at least partially address aspects of the feature summarized by Topic 46. This is particularly relevant in the context of risk-based testing where the focus is set to quality assure a particular functionality that exhibits high risk exposure.

Another potential application of the information provided by topic models, not explored in this chapter, is test case prioritization. The test cases in Figure 7.1 contain context dependent information, e. g. objective, initialization and approval criteria for the test case. To support prioritization, one could check initialization dependencies through topic associations. For example, the initialization field in test case #293 is associated with Topic 11 which could indicate that executing test case #335 before #293 would reduce the test effort in case #335 reveals a fault. We have left such extension to future work.

LDA requires configuration through a set of parameters [Blei et al., 2003, Panichella et al., 2013]. The particular values for these parameters depend on the task at hand, which in turn determines how the quality of a topic model should be evaluated. For example, one LDA parameter is the number of topics of the model, affecting the interpretability of the results [Steyvers and Griffiths, 2007]. Choosing a too low number may create coarse grained topics while the other extreme may lead to topics that are overly specific. Evaluating the quality of topic models is an area of active research in general (e. g. topic generalizability [Blei et al., 2003, Wallach et al., 2009b], and topic coherence and interpretability [Chang et al., 2009, Newman et al., 2010, Mimno et al., 2011, Lau et al., 2014]), but also specifically for software engineering tasks (e. g. Hindle et al. [2014] and this chapter).

<sup>1</sup> Note that each document is associated with a distribution over all topics in the model. We show in this example only topics with a probability > 1%.

## 2.4 Topic Models in Software Engineering

Topic models have been used in a large variety of applications in software engineering, e.g. program comprehension [Maskeri et al., 2008, De Lucia et al., 2013], recovering of traceability links [Asuncion et al., 2010, Nguyen et al., 2011], impact analysis [Gethers and Poshyvanyk, 2010], and the evolution of source code [Thomas et al., 2011] and requirements [Galvis Carreño and Winbladh, 2013]. Here, we focus our discussion on the use of topic models relevant to our proposed approach for test case selection described in Section 3.

Thomas et al. [2014] use the linguistic data present in executable test cases, e.g. string literals, comments and identifier names, to create topics models. They use the created topic model to differentiate the tested functionality and to prioritize test cases, maximizing the topics they cover. The results indicate that their approach is as effective as dynamic approaches based on call graphs while still being inexpensive to implement.

While Thomas et al. [2014] focused on maximizing coverage, Hemmati et al. [2015] compare three different strategies for test case prioritization: covering a maximum amount of topics, maximizing test diversity and considering historic failure detection performance. In contrast to Thomas et al. [2014], the test cases are not executable but manual test cases that contain test steps and acceptance criteria, rendering prioritization even more important as their execution is time intensive. Their results indicate that, in the context of traditional software development, none of the prioritization strategies predominates while in a rapid release environment, the risk-driven strategy is superior.

## 2.5 Contribution

Our application and use of topic models in test case selection is novel in two aspects. First, in our approach, the engineer uses directly the topic model, exploiting its mixed-membership nature. Similar studies to ours, looking for example at test case prioritization [Thomas et al., 2014, Hemmati et al., 2015], use topic models to calculate similarities and to automatize decisions like test case prioritization. Our approach integrates the engineer in the decision process, addressing out-of-the-loop unfamiliarity [Endsley and Kaber, 1999]. The out-of-the-loop problem refers to the performance drop of engineers reverting back to a manual process when an automated process fails, leading to a lack of trust in automation [Parasuraman et al., 2000]. Second, we evaluate the intrinsic *and* extrinsic quality [Mimno et al., 2011] of topic models. The intrinsic evaluation refers to model selection, which we automate, using recent ideas in this area [Lau et al., 2014], design and optimize for the task at hand, i.e. the extrinsic eval-

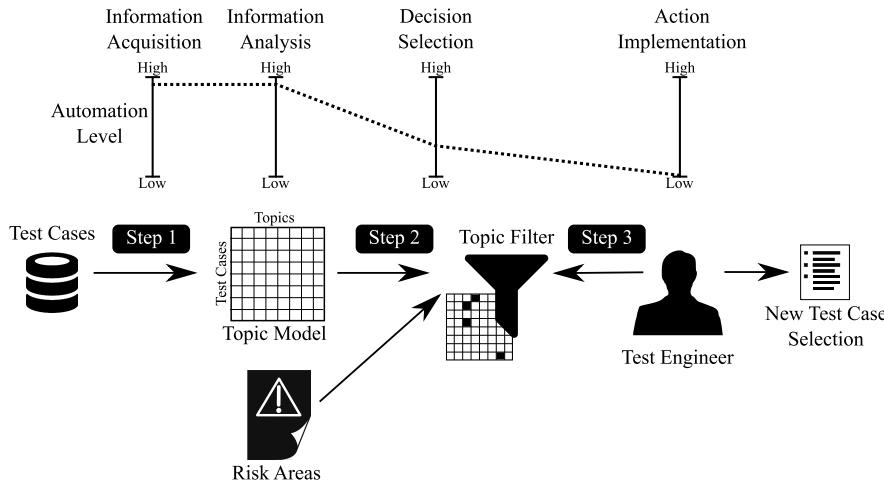


Figure 7.2: Risk-based Testing supported by Topic Models (RiTTM), with corresponding levels of automation [Parasuraman et al., 2000]

uation. This is in contrast to work (e.g. Hindle et al. [2014]) that performs a subjective and manual intrinsic evaluation to select a topic model for the task at hand.

On a wider perspective, our results in model selection corroborate the observations by Chang et al. [2009] who found that model perplexity and interpretability were negatively correlated on their data set. This underpins the importance of choosing the appropriate evaluation metric for model selection, depending on whether the focus lies on generalizability or interpretability of the topic model.

### 3 RISK-BASED TESTING SUPPORTED BY TOPIC MODELS (RITTM)

In this section we describe our approach to support engineers in system test case selection using topic models. It is important to point out that our goal is not to automatize test case selection but to provide decision support for test engineers. Other approaches, for example using feature models for automated test case selection [Wang et al., 2013] aim at reducing the reliance on domain expertise. However, this comes at the cost of investing in infrastructure (feature model, establish and maintain traceability between feature model and test cases) that may not exist or not be stable enough to enable an automated approach.

Our goal is to increase the confidence of domain experts in their test case selection by providing decision support originating from the analysis of their test case database. This goal is in line with the software analytics principles [Menzies and Zimmermann, 2013], in particular with the aspect that the data min-

ing process should be driven by the need of the user. Figure 7.2 provides a schematic outline of RiTTM, showing artifacts that we use as input (test cases and risk areas) to create a topic model and a topic filter that supports a test engineer to perform test case selection (on top in the figure we also show automation levels, more on them below). The main idea is to create a topic model that provides an alternative view of the test cases which are typically grouped into a specific hierarchy or classification (Step 1). The test engineer uses then the topic model in combination with risk area specifications to create a topic filter (Step 2), which he uses then to perform test case selection (Step 3). When developing a system that interacts with humans, it is also important to consider the consequences of automation on the users of the system [Parasuraman et al., 2000]. On the top in Figure 7.2 we show the levels of automation [Parasuraman et al., 2000] we have chosen (or obtain) for each step. In Step 1, which combines information acquisition and analysis, we chose a high level of automation since this can be implemented reliably and cost effective, reducing the mental workload of the user. In Step 2 we reduce the automation, narrowing down the selection with the topic filter, keeping however the human in charge in taking decisions. Action implementation, i. e. actually selecting test cases and executing them is not automated and not in the scope of our approach; it can be partly or fully automated but that decision is orthogonal to the use of RiTTM.

RiTTM does not optimize the test selection w.r.t. coverage and execution time but increases the basis on which a test engineer can make an informed decision. Therefore, the goal of RiTTM is to be serendipitous [Avazpour et al., 2014], i. e. to facilitate the discovery of new and relevant test cases, rather than to replicate the selection decisions by a test engineer. André et al. [2009] suggest that it is possible to design serendipity into systems by automating and accelerating the discovery of information while facilitating the human user in the process of making connections in that information. This design principle is reflected in RiTTM, where the information acquisition and analysis process has a high level of automation while decision selection is supported by suggestions and still under full control of the user (see Figure 7.2).

We have developed RiTTM in the context of a company which performs risk-based testing (see Section 4.1), and designed therefore the test case selection process within the framework given by the engineers at the case company. We assume that some sort of risk exposure assessment exists for product features that ought to be tested. However, the use of risk specifications is one example of information that can be used to create the topic filter; alternatives could be a description of changed features or requirements. In the remainder of this section, we explain how to implement the individual steps in our proposed test case selection process.

### 3.1 Step 1 – Model Selection

Model selection is the process of configuring an algorithm with optimal parameters, evaluated by an objective performance function [Lavesson and Davidsson, 2006, Japkowicz and Shah, 2011]. However, “the unsupervised nature of topic models makes model selection difficult” [Wallach et al., 2009b]. Basically, and in contrast to supervised models, there is no “correct” number of topics to which one can compare and evaluate results. In this section we discuss the parameters that can be set in LDA and the performance functions that can be used to evaluate the quality and steer the selection of a topic model.

#### 3.1.1 Which Parameter to Optimize?

LDA is configured by 3 hyper-parameters<sup>2</sup>,  $\alpha$ ,  $\beta$ , and  $T$ , that influence the smoothing effect on the generated topic model [Griffiths and Steyvers, 2004]. The  $\alpha$  prior influences the topic distributions per document. Setting  $\alpha$  to a large value expresses our prior belief that all topics tend to be distributed evenly in every document. On the contrary, a small  $\alpha$  expresses our belief in a distribution where only a few topics exhibit a high proportion in each document. Similarly, the  $\beta$  prior influences the terms’ distribution per topic. Note that even though the priors are important if there is only little data on which to train the topic model their influence is less critical when there is more data. This is because topic models are at heart Bayesian generative models where the prior beliefs are updated based on the actual evidence.  $T$  is the number of topics and determines the granularity of the topic model and is related to the interpretability of the individual topics [Steyvers and Griffiths, 2007]. We optimize  $\alpha$  and  $\beta$  using the approach by Wallach et al. [2009a] to hyper-parameter updating, setting the starting values to  $\alpha = 50/T$  and  $\beta = 0.01$  which have been proven useful for natural language texts [Steyvers and Griffiths, 2007]. Consequently, we vary only  $T = 5..300$  and evaluate the quality of the generated models according to the performance measures discussed next.

#### 3.1.2 How to evaluate the quality of the topic model?

Topic models have been evaluated traditionally by measuring perplexity, estimating the models’ generalization performance [Hofmann, 1999, Blei et al., 2003, Griffiths and Steyvers, 2004, Wallach et al., 2009b]. The document collection is split into a training and a test set. Perplexity then measures the uncertainty of predicting a single term from the test set, given

<sup>2</sup> If LDA is implemented with Gibbs sampling [Griffiths and Steyvers, 2004], the number of burn-in iterations, the number of samples and the sampling interval are additional parameters that can be set [Binkley et al., 2014]. We use Gibbs sampling, however do currently not optimize these parameters.

the term distribution in the trained model [Griffiths and Steyvers, 2004]. However, such a measure is useful for evaluating the predictive nature of the model, but do not address the exploratory goal of topic models [Chang et al., 2009]. Step 2 in our approach (Figure 7.2) requires a human to interpret the model in the context of risk areas to create the topic filter. For this task, the topic model should be interpretable rather than generalize to new test cases. Chang et al. [2009] propose a topic quality evaluation based on topic coherence. Their basic idea is to measure whether a human can detect an intruder in the set of terms that define a topic. In a coherent topic, a human is more likely to identify a term that is unrelated to all other terms, while in an incoherent topic, this task is more difficult. The experiment conducted by Chang et al. [2009] reveals a surprising result: model perplexity is negatively correlated with topic coherence. In other words, as topics become more fine-grained and numerous, they are less useful to humans [Chang et al., 2009]. However, to implement the topic quality metric proposed by [Chang et al., 2009], human judgment is needed which is not feasible for the many possible models involved in the model selection step. Therefore, automated methods without human involvement have been proposed, e. g. by evaluating semantic similarity of terms using ontologies, encyclopedias and search engines [Newman et al., 2010], by exploiting co-occurrence information of terms [Mimno et al., 2011], and by clustering terms from topic model instances to determine topic stability [Mehta et al., 2014].

We chose to implement the approach by Lau et al. [2014] to automatically measure topic coherence by term intrusion. They extend the work by Chang et al. [2009] by replacing the human judge of topic coherence with a machine learner that reaches a decision based on word associations measures learned from a larger document collection. We illustrate the implementation of model selection in Section 4.2.1.

### 3.2 Step 2 – Topic Filter Creation

The test engineer creates the topic filter based on the identified risk areas for the particular test project. A risk area specification is a document that associates a risk item with a risk exposure score. Depending on the organization's level of documentation and the purpose of the tests, risk exposure can be assigned to different risk items that represent product features, e. g. requirements specifications, architecture and design documents or test cases [Felderer and Schieferdecker, 2014]. For example, if an up-to-date and complete requirements specification exists, product managers and developers could assign, depending on the importance of a requirement and the complexity of the implementation, assign a risk exposure score to each requirement, using e. g. the process proposed by Redmill

| Testcase name    | Topic Level 1 |            | Topic Level 2 |            | ... | Topic Level N |            |
|------------------|---------------|------------|---------------|------------|-----|---------------|------------|
|                  | ID            | Proportion | ID            | Proportion |     | ID            | Proportion |
| Functionality A1 | 0             |            | 24            |            | ... | 29            |            |
| Functionality A2 |               |            | 31            |            |     | 15            |            |
| Functionality D2 |               |            | 5             |            |     | 8             |            |
| Functionality F3 | 1             |            | 17            |            | ... | 44            |            |
| Functionality A3 |               |            | 2             |            |     | 18            |            |
| Functionality K1 | 2             |            | 7             |            | ... | 45            |            |
| Functionality K2 |               |            | 29            |            |     | 22            |            |
| Functionality K5 |               |            | 0             |            |     | 41            |            |
| :                | :             | :          | :             | :          | :   | :             | :          |

Figure 7.3: Topic filter table implementation

[2005]. In Section 4.2 we illustrate how this process works at the case company.

The risk area specification serves then as input to select relevant topics from the previously created topic model. By relevant we mean whether the concept described in the topic can be associated with the risk area(s) described in the specification or that are currently in focus. We designed this process deliberately to be manual, involving the test engineer to actively use his domain knowledge to integrate topics with risks. This also explains why we select models with high coherence rather than for their potential to generalize to a hold-out, test set. Once risks are associated with topics, filtering of test cases can take place by considering the topic proportions of each test case and sorting/arranging the information in the most convenient way. Figure 7.3 shows one possible approach to implement the topic filter, i. e. in a table. Each row shows a test case and its corresponding composition in terms of topics from the model. The test cases are grouped according to their main topic (Topic Level 1), in decreasing order of proportion w.r.t. that topic. The second level shows the topic with the second largest proportion for each test case, and so on until Topic Level N. Since we parameterized LDA with a small  $\alpha$  (see Section 3.1), the algorithm tends to produce topic models with few, high proportion topics [Tang et al., 2014]. This is a property we expect and prefer, since test cases are likely to contain a few (1–4) main concepts/features. Hence, the table in Figure 7.3 should have typically 3–4 topic levels until the proportion falls below 1%.

### 3.3 Step 3 – Test Case Selection Decision Support

Test case selection is now a matter of using the topic filter, together with the chosen topics matching risk areas, to select individual test cases. A sensible implementation would also filter out test cases that have already been selected by the test engineer. Using the topic filter shown in Figure 7.3, let us assume the test engineer associated topic ID = 0 with a particular risk area, then the filter would suggest the tester to inspect test cases A1, A2 and D2 (on Topic Level 1), and test case K5, based on its proportion for topic with ID = 0 on Topic Level 2. The test engineer can then decide whether the suggested test cases should be added to the selection.

Our conjecture is that test case selection supported by topic models can suggest test cases the engineer would not have thought of including or simply forgot. We investigate this hypothesis in a case study, illustrated in the remainder of this chapter.

## 4 CASE STUDY DESIGN

Our overall goal in this case study is to answer the question *to what extent can RiTTM suggest new and relevant test cases?*

With *new* test cases we do not mean to generate test cases that did not exist before, but test cases that have not yet been considered by an engineer in the test case selection while still being relevant. The reasons for not including relevant test cases can lie in time or resource constraints, both in the selection and in the execution process. RiTTM is *not* designed to identify a minimal set of test cases, optimizing the test effort. We assume that the goal of the selection is to achieve higher test coverage of those areas that were assigned a high risk exposure. This is the use case for RiTTM that we have elaborated together with ES, our case company. The case context is further described in Section 4.1. We illustrate the implementation and execution of RiTTM in Section 4.2, following the design decisions described in Section 3. To evaluate the performance of RiTTM, the author of this chapter replicated a test selection supported by topic models and compared the result with the selection performed by a test engineer. The evaluation is further described in Section 4.3. Threats to validity are discussed in Section 4.4.

### 4.1 Case Context

Company ES (“Embedded Systems”, name anonymized for confidentiality reasons) develops both hard- and software for their worldwide marketed products. ES has a three-decade history in developing embedded systems, although the particular applications areas have changed over time.

At the beginning of our collaboration with ES, we performed a lightweight process assessment [Pettersson et al., 2008] in order to identify improvement opportunities, in particular in the coordination between requirements engineering and software testing. We interviewed 16 employees (line managers, product and project managers, test and technology leads, and test engineers) and reviewed project documentation. We identified a lack of variability management in the problem space (requirements analysis, specification and maintenance), which is a rather common situation. Chen and Babar [2010] report that challenging extraction of variability from technical artifacts, maintaining variability documentation over time and tool support are detractors for successful variability management in industry. Similarly, Thörn [2010] observed a tendency towards problem descriptions that are exclusive for each product. This can then lead to shortcomings in internal communication and an increased risk of duplicated work [Thörn, 2010]. At ES, the reduced focus on variability management in the problem space has historical reasons that can be attributed to the strong technical support for managing variants in the solution space (separate development of a platform and product specific software). ES generally develops a new product generation by reusing and extending the existing requirements specifications. This leads to a lightweight requirements management process, but also to challenges for impact analysis of new features since relationships between requirements are not documented [Graaf et al., 2003].

ES continually runs programs to improve their quality assurance efficiency. They introduced automated integration tests that are run on every version control system commit and before the product is handed over to the quality assurance (QA) department. Within our collaboration, we have conducted a series of experiments that aimed at re-engineering traceability between source code and test cases using information retrieval techniques such that test case selection can be based on the actually configured product (see Chapter 6). The results indicated that the performance of the chosen techniques depends on the degree to which a configuration change affects the source code, and thereby the signal that can be used to distinguish between test cases. Given the reported challenges in Chapter 6, we reoriented our research towards supporting ES in their risk-based testing strategy which they developed to focus effort on those parts of the product carrying a risk to contain faults, optimizing thereby available time and resources. We now outline the process at ES that we aim to support with the approach presented in Section 3. Figure 7.4 illustrates the risk-based testing strategy applied for every newly developed product whose new features are integrated into the firmware platform underlying all products. Risk areas and test cases are organized according to system test areas, thereby tracing risks to a particular set of test cases. The risk exposure

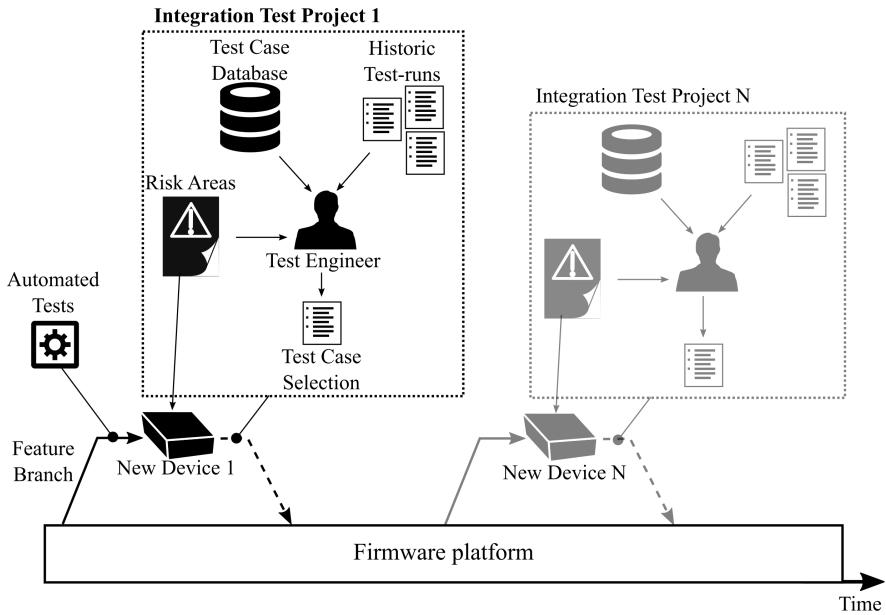


Figure 7.4: Testing process for new product development and integration into the firmware platform

score for each system test area, on a five point scale between none and very high, is determined by expert judgment originating from:

- product managers, estimating the cost if a feature fails in the field
- developers, estimating the likelihood that a system test area is affected by feature changes
- test engineers, estimating the likelihood that a system test area detects faults, based on previous test-runs

The risk area specification is created in a collaborative meeting between the above listed roles, recording also details on changed features and which product groups and firmware versions are affected. This document, together with historic test-runs on similar products, form the decision basis on which the test engineer performs the selection for the integration test project. The test case database at ES contains approximately 1500 manual system test cases, rendering the risk-based selection paramount to competitive time to market while maintaining high product quality and reliability.

#### 4.2 *Implementation of RiTTM*

While we described RiTTM and motivated design decisions in Section 3, we focus in this section on a reference implementation that is targeted at

evaluating the feasibility of the approach within the industrial context illustrated in Section 4.1. We describe how we implemented the steps shown in Figure 7.2. All three steps were performed by the author of this chapter; a test engineer from ES was involved in the evaluation as described in Section 4.3.

We chose an integration test project that was recently completed (summer 2014) at the time when we conducted the case study. The risk area specification of that project contained 87 system test areas, corresponding to test modules in the test case database, each of which was assigned a risk exposure between 0 (none) and 5 (very high). For each system test area with an exposure higher than 0, the integrated feature is described and a motivation why the particular risk area is affected is provided. In this project, five system test areas were assigned a high risk exposure and were therefore selected for the case study. The original selection performed by the test engineer contained 111 system test cases, of which 55 belong to the test modules assigned a high risk exposure.

#### 4.2.1 Step 1 – Model Selection

Since test cases are central in the quality assurance process, ES continually improves their test case management. Besides scheduling, effort estimation and task assignment, the test management software traces discovered faults to test cases, provides test case versioning and assigns ownership to test case maintainers. Each test case, assigned to a system test area that groups test cases related to a particular product feature, consists of an overall test objective, initialization information, assumptions, approval criteria and a set of test steps. This test case content is English natural language text. We extract this content from the SQL database and feed it into Mallet [McCallum, 2002], a tool set for natural language processing and machine learning. We use Mallet to pre-processes the text: special characters, numbers and stop-word removal, lower case transformation, and tokenization. These are standard steps in natural language processing [Falessi et al., 2013]. For the model selection proper, we use Mallet to train 60 models varying the number of topics ( $T = 5, 15, 20, \dots, 300$ ) with 1000 Gibbs sampling iterations, setting the initial values for  $\alpha = 50/T$  and  $\beta = 0.01$  and letting Mallet optimize [Wallach et al., 2009a] these two parameters with an optimization interval of 10 and optimize burn-in of 20 samples (note that these are Mallets' suggested default values). We repeat this process 10 times, with a different random seed, resulting in a total of 600 topic models. To perform model selection, we calculate model precision, a measure proposed by Chang et al. [2009] and implemented by Lau et al. [2014]. However, we adapt their implementation in two aspects:

1. We sample the intruder term, i. e. the term that is inserted into the list of terms defining a topic and needs to be detected, from the high

probability terms of a random topic in the model instead from the whole document collection. The motivation for this is to avoid making the task of detecting the intruder term too easy because the term has an overall very low probability [Chang et al., 2009].

2. To sample lexical probabilities required to compute the term association feature<sup>3</sup>, we use the complete system test case database that contains, software tests, tests that verify mechanical and electronic properties of a device and legacy tests that apply to previous product generations. Lau et al. [2014] used New York Times News and Wikipedia articles.

One could now select the “best” model by simply choosing the one with the highest model precision. However, we argue that the choice of  $T$ , the number of topics, should not be purely based on model precision, but consider also the task at hand. Thus, we use the following criteria for model selection:

1. Model precision: ranges between 0..1 and expresses how many term intruders were determined correctly;  $T$  with higher values of model precision are preferable.
2. Model precision variance: since LDA, in particular Gibbs sampling, has a component of randomness [Steyvers and Griffiths, 2007], repeated runs with exactly the same parameters lead to slightly different topic models;  $T$  with lower variance of model precision are preferable.
3. Model granularity: the task at hand and the actual data influence the choice for  $T$  and thereby the granularity of the model.

The results of model selection are presented in Section 5.1.

#### 4.2.2 Step 2 – Topic Filter Creation

Once the topic model is selected, we represent the model in a spreadsheet, as illustrated in Figure 7.3. Rows contain test cases while columns indicate the topic association and proportion to each test case. This allows us to filter out test cases that are associated to a certain topic, identified by an ID. Each topic consists of 15 terms determining the meaning of the topic.

We now have to determine how many levels the topic filter should include. In LDA, each document is associated with a distribution over all topics in the model; however, we tuned LDA to produce a model where only a few topics have a high probability, as motivated in Section 3.2. Figure 7.5 shows the percentage of test cases, at different topic levels, where the topic proportion  $> 1\%$ . For example, at topic level 2, 82.8% of the test

---

<sup>3</sup> Term association refers to a measure expressing the likelihood of observing the co-occurrence of terms. We use normalized pointwise mutual information as suggested by Lau et al. [2014].

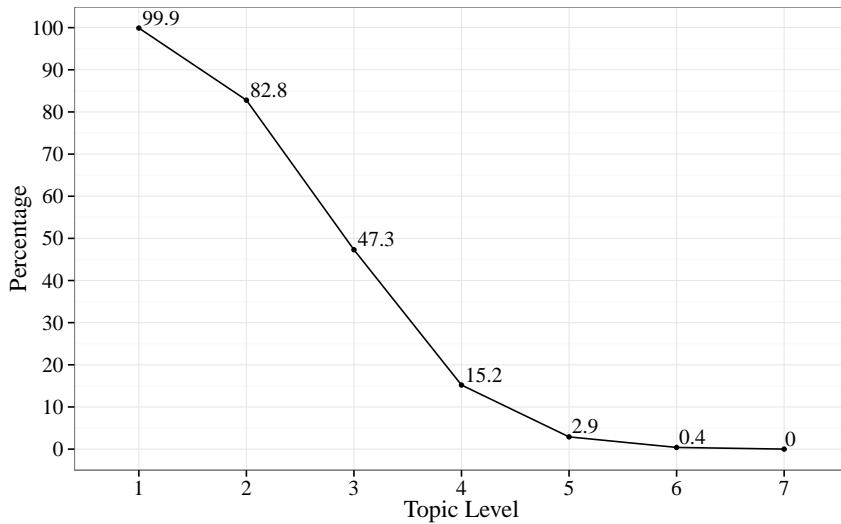


Figure 7.5: Test case percentage where topic proportion  $> 1\%$  at different topic levels

cases exhibit a topic distribution where the 2nd largest topic proportion is larger than 1%. This analysis provides an indication for a reasonable cut-off point: we chose topic level 4 since the next level (5) would expand the set of potential test cases only by 2.9%.

Next, the user of the topic filter needs now to integrate the information from the topic model and the selected risk areas (see Step 2 in Figure 7.2) as follows:

1. Choose a risk area where the risk exposure is above a predetermined threshold. We set the threshold to 4 (high on the company internal 5-point scale).
2. Read the risk area description, the description of the integrated feature and the motivation why this risk area is affected.
3. Read the terms of each topic and mark those who can be associated to the information gathered in 1. The association can be based on similarity of terms, or, and this is the reason why we do not simply use a similarity measure between risk area specification and topic terms, based on semantic association of the topic as a whole or individual terms with the risk area. For example, if the risk area describes a feature related to audio capabilities, there is a wide range of concepts that can be associated with audio: wave, microphone, speaker, acoustic, quality, distortion etc. We rely here on the human capability and domain knowledge to create these associations and connect risk areas to topic terms stemming from test cases.

4. If all risk areas above the risk exposure threshold are considered, stop. Otherwise, continue with the first step.

The result is a list of risk areas, each mapped to one or more topics.

#### 4.2.3 Step 3 – Test Case Selection Decision Support

Performing test case selection is now a matter of applying the topic filter for each risk area and deciding whether the resulting test cases are relevant, again based on the information provided in the risk area specification. For each topic, the first author inspected the test cases at topic level 1. If a relevant test case could be identified at topic level 1, the test cases that had the same topic at lower topic levels were inspected too. Note that by “relevant” we mean according to the understanding of the author of this chapter who is not an expert in the domain of the tested product. Next we describe the design of the test case selection evaluation.

### 4.3 Evaluation of RiTTM

As described in Section 4.2, we chose a recent integration test project for the implementation of RiTTM. In the following, we use the term “expert” to refer to the test engineer at ES who performed the original selection, while we use the term “non-expert” to refer to the first author. We benchmark the test case selection, performed by the expert, against the test case selection by the non-expert using RiTTM. For each risk area, we classified the number of test cases as follows (the letters in bold typeface refer to the columns in Table 7.1 which shows the results of the classification):

**Total selected:** number of test cases that were associated with the corresponding topic (at topic levels 1–4).

**t****opic selected:** number of test cases that were selected by the non-expert as being relevant for the particular risk area.

**Both selected:** number of test cases that were chosen by both the expert and the non-expert.

**new and Relevant:** number of test cases that were only selected by the non-expert and were judged as relevant by the expert.

**Not applicable:** number of test cases that the non-expert selected, but are not applicable according to the expert since the product under test or the firmware does not have the tested capability.

**Unrelated:** number of test cases that the non-expert selected, but are not relevant according to the expert.

We quantify the *efficiency* of RiTTM by the percentage of the test cases that were filtered out from the total number of test cases (1415). In other words, the filter efficiency expresses the percentage of test cases that the

non-expert did not have to inspect. Furthermore, we quantify the *coverage* of RiTTM by the number of test cases that the non-expert selected out of the ones that were originally selected by the expert. Finally, we quantify the *serendipity* of RiTTM, i.e. its capability to discover new, relevant and unexpected test cases, by the number of test cases the non-expert selected using RiTTM, that were not sorted in the test module that corresponded to the risk area specification<sup>4</sup>.

#### 4.4 Validity Threats

We designed our case study in an industry environment to evaluate both feasibility and scalability of RiTTM. Nevertheless, we note the following internal and external validity threats.

##### 4.4.1 Internal Validity

This category of threats refers to the validity of the observed causal relations [Runeson and Höst, 2009]. We designed RiTTM to incorporate a pragmatic topic model selection process. While we took great care to choose an appropriate evaluation metric, model precision, the interpretability of a topic is still evaluated automatically and is not verified by a human. Even though Lau et al. [2014] showed that this approach leads to results that correlate with human judgment, our data set differs in that its domain is very specialized. Nevertheless, our results from the case study indicate that the selected model was indeed useful for the task at hand. Examining whether a topic model with more or fewer topics would lead to significantly different results would require the setup of a controlled experiment, which was not the purpose of this study.

We set the number of terms per topic to 15 and do not vary this parameter in model selection. This number was chosen on the intuition originating from preliminary models where we experimented with 10, 15 and 20 terms. While we think that 15 terms is a lower bound, further experiments are necessary to determine an optimal trade-off between the information a topic conveys and its interpretability. None of the studies on topic coherence [Chang et al., 2009, Newman et al., 2010, Mimno et al., 2011, Lau et al., 2014] addresses this issue.

##### 4.4.2 External Validity

This category of threats refers to the extent to which the findings of a study can be generalized [Runeson and Höst, 2009]. Within the context of ES, we

<sup>4</sup> Remember that we discussed in Section 4.1 that the risk area specification is organized according to test modules in order to achieve traceability and facilitate test case selection for the test engineer.

consider the selected integration test project as typical. The results from the studied five risk areas can therefore be extrapolated to all integration projects.

We designed the evaluation without the active use of RiTTM by a test engineer from ES since the handling of the topic filter (implemented as a spreadsheet) is not intuitive and may distract from the task. The researcher using RiTTM collaborated over a period of three years with ES, acquiring some understanding of the product domain without however reaching the level of the test engineers employed at ES. The achieved results are therefore even more encouraging since RiTTM may even provide support for non-experts in test case selection.

With respect to the external validity outside the context of ES, we expect that our proposed approach generalizes to other companies that perform system test case selection. There are limitations on the document collections on which topic models perform well [Tang et al., 2014], which may influence the performance of RiTTM.

## 5 CASE STUDY RESULTS

In this section we present and discuss the results of the intrinsic quality evaluation of the topic model selection (Section 5.1) and the extrinsic quality evaluation of the task of test case selection supported by a topic model (Section 5.2).

### 5.1 Model Selection Results

Figure 7.6 shows a box-plot diagram over 10 iterations for model precision for 60 topic models where  $T = 5..300$ . Without considering how the topic model will be used, one would pick the model with the highest precision ( $T = 20$ ). However, this would mean for the topic filter that each topic is associated with, on average, 71 test cases (the database contains 1415 test cases). This renders the filter rather coarse grained and less useful. Furthermore, there is a large variance in precision for models with  $T = 5..50$ . Model precision stabilizes at  $T > 60$ , however decreases at the same time too. Given the three criteria we specified before, any  $T = 60..80$  would be a reasonable choice. We chose  $T = 70$  and created the topic filter with 70 topics.

#### 5.1.1 Discussion

Model precision expresses how coherent or interpretable, on average, the topics in a model are. A model with fewer topics results in broader topics [Steyvers and Griffiths, 2007]. Since we sample intrusion terms from high

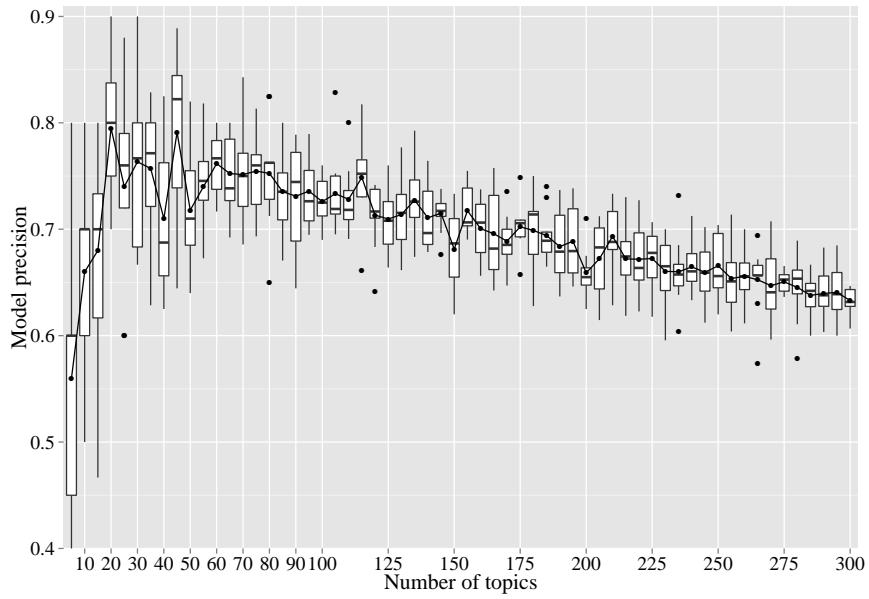


Figure 7.6: Model precision of 600 topic models resulting from the procedure detailed in Section 4.2.1

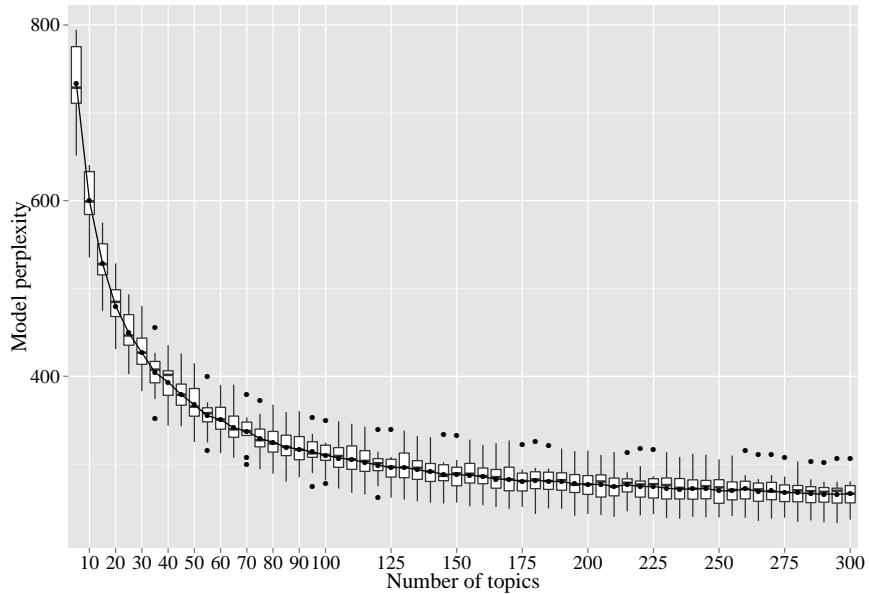


Figure 7.7: Model perplexity of 600 topic models

probability terms in other topics of the model, broad topics are not only penalized by a low model precision (for  $T \leq 15$ ), but also by a large model precision variance (see Figure 7.6 where  $T \leq 45$ ). For test case selection, we prefer topics that are not broad, i. e. that are orthogonal (do not overlap in their semantics) to each other. This reduces the number of test cases that are associated to a topic, making the topic filter more efficient. On the other hand, a model with too many topics is difficult to use since each topic needs to be judged whether it is associated to a risk area while topic interpretability decreases (see Figure 7.6 where  $T > 80$ ). When consciously considering these trade-offs and the particular use case for the model, we think that model precision is a good choice for a topic model selection metric.

To further substantiate this claim, we also calculated model perplexity [Blei et al., 2003], a measure that evaluates the generalizability of a topic model. We vary again  $T = 5, 10, 15, \dots, 300$ , performing a 10-fold cross-validation, measuring the model perplexity on the hold out test cases. Figure 7.7 shows a box-plot diagram of the results. Using the same reasoning as with the model precision metric, one would probably choose a topic model where  $T > 40$ . Then, however, it is much more difficult to justify a particular choice for  $T$  since perplexity continues to decrease.

Looking at Figure 7.6 and 7.7 we can also observe that the model precision and perplexity measures correlate (Spearman's rank correlation coefficient of the means of model precision and perplexity is 0.76). However, note that lower model perplexity is better, while for model precision, a higher value is better. This corroborates Chang's et al. observations that topic models which exhibit better generalization performance may produce less interpretable topics [Chang et al., 2009].

## 5.2 Test Case Selection Results

As a result of the topic filtering process, described in Section 4.2.2, we chose 22 out of 70 possible topics for the given five risk areas. Following the process described in Section 4.2.3, this led to the identification of 274 test cases which were then judged by the test engineer who did the original selection at ES. Table 7.1 summarizes the results of the experts assessment. The bottom of the table shows the filter efficiency, coverage and serendipity for each risk area.

Filter efficiency is the percentage from the total of the 1415 test cases that were filtered out by the topic model. This number shows the reduction of effort spent, compared to inspecting all test cases, to achieve a test case selection result expressed by coverage and serendipity, which are effectiveness measures. Over the five risk areas, RiTTM achieved a filter efficiency

Table 7.1: Test case selection results. The abbreviations in the columns refer to test cases: Total selected, tTopic selected, Both selected, new and Relevant, Not applicable, Unrelated

| Topic             | T  | O      | Risk Area 1 |         |     | Risk Area 2 |      |         | Risk Area 3 |        |      | Risk Area 4 |     |        | Risk Area 5 |         |                 |         |        |         |         |
|-------------------|----|--------|-------------|---------|-----|-------------|------|---------|-------------|--------|------|-------------|-----|--------|-------------|---------|-----------------|---------|--------|---------|---------|
|                   |    |        | R           | B       | N   | U           | T    | O       | B           | R      | N    | U           | T   | O      | B           | R       | N               |         |        |         |         |
| T5                | 40 | 7      | 3           | 1       | 3   | 0           | 36   | 7       | 4           | 1      | 2    | 0           |     |        |             |         |                 |         |        |         |         |
| T38               | 35 | 0      | 0           | 0       | 0   | 0           | 19   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T12               |    |        |             |         |     |             | 43   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T22               |    |        |             |         |     |             | 11   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T37               |    |        |             |         |     |             | 14   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T39               |    |        |             |         |     |             | 37   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T55               |    |        |             |         |     |             | 73   | 7       | 5           | 0      | 2    | 0           |     |        |             |         |                 |         |        |         |         |
| T61               |    |        |             |         |     |             | 27   | 0       | 0           | 0      | 0    | 0           |     |        |             |         |                 |         |        |         |         |
| T67               |    |        |             |         |     |             |      | 94      | 23          | 3      | 3    | 8           | 9   |        |             |         |                 |         |        |         |         |
| T17               |    |        |             |         |     |             |      | 60      | 22          | 2      | 2    | 12          | 6   |        |             |         |                 |         |        |         |         |
| T29               |    |        |             |         |     |             |      | 36      | 5           | 2      | 0    | 2           | 1   |        |             |         |                 |         |        |         |         |
| T31               |    |        |             |         |     |             |      | 43      | 26          | 8      | 2    | 12          | 4   |        |             |         |                 |         |        |         |         |
| T69               |    |        |             |         |     |             |      |         | 25          | 12     | 0    | 0           | 4   | 8      |             |         |                 |         |        |         |         |
| T27               |    |        |             |         |     |             |      |         | 35          | 20     | 6    | 1           | 9   | 4      |             |         |                 |         |        |         |         |
| T46               |    |        |             |         |     |             |      |         | 18          | 14     | 2    | 2           | 9   | 1      |             |         |                 |         |        |         |         |
| T58               |    |        |             |         |     |             |      |         | 31          | 3      | 0    | 1           | 1   | 1      |             |         |                 |         |        |         |         |
| T60               |    |        |             |         |     |             |      |         | 90          | 36     | 11   | 9           | 10  | 6      |             |         |                 |         |        |         |         |
| T64               |    |        |             |         |     |             |      |         | 19          | 1      | 1    | 0           | 0   | 0      |             |         |                 |         |        |         |         |
| T68               |    |        |             |         |     |             |      |         |             | 108    | 28   | 5           | 6   | 3      | 15          |         |                 |         |        |         |         |
| T8                |    |        |             |         |     |             |      |         |             | 20     | 7    | 0           | 0   | 4      | 3           |         |                 |         |        |         |         |
| T28               |    |        |             |         |     |             |      |         |             | 162    | 52   | 6           | 1   | 8      | 37          |         |                 |         |        |         |         |
| T47               |    |        |             |         |     |             |      |         |             |        |      |             |     |        |             |         |                 |         |        |         |         |
| Sum               | 75 | 7      | 3           | 1       | 3   | 0           | 260  | 14      | 9           | 1      | 4    | 0           | 233 | 76     | 15          | 7       | 34              |         |        |         |         |
| Filter efficiency | 75 | out of | 1415        | (94.7%) | 260 | out of      | 1415 | (81.6%) | 233         | out of | 1415 | (83.5%)     | 218 | out of | (84.6%)     | 290     | out of          | (97.5%) |        |         |         |
| Coverage          | 2  | out of | 13          | (15.4%) | 9   | out of      | 15   | (60.0%) | 7           | out of | 8    | (87.5%)     | 15  | out of | 16          | (93.7%) | 2               | out of  | 3      | (66.6%) |         |
| Serendipity       | 0  | out of | 1           | (0%)    | 0   | out of      | 1    | (0%)    | 5           | out of | 7    | (71.4%)     | 4   | [8]    | out of      | 13      | (30.8% [61.5%]) | 4       | out of | 7       | (57.1%) |

between 79.5% and 94.7%, a coverage between 15.4% and 93.7% and a serendipity between 0% and 71.4%.

### 5.2.1 Discussion

Looking at the distribution of topics among risk areas (Table 7.1) we can observe that only topic T5 was associated with more than one risk area. This is an indication of the extrinsic quality of the selected topic model. This orthogonality of topics is important for filter efficiency; if topics would overlap, more topics would be associated to risk areas, increasing the number of test cases that need to be inspected. Filter efficiency can be further improved if not relevant test cases would be preemptively removed from the pool of selectable test cases. However, the necessary information to implement this improvement is not available in the test case database at ES. Currently, test engineers need to check historic test runs or with product experts whether a test case is relevant for a particular product model or firmware version.

The coverage figures in Table 7.1 seem to indicate that RiTTM's strength is not to replicate a test case selection performed by an expert. Nevertheless, considering that RiTTM was applied by a non-expert in the product domain, the coverage is reasonable (between 15.4% and 93.7%, depending on the studied risk area). We expect that coverage increases considerably if RiTTM is applied by a domain expert since 19 out of 20 test cases that the non-expert did not select were actually in the filtered set produced by RiTTM. We assume that a domain expert would have selected them, reaching with RiTTM a coverage of 98.2% of the originally selected test cases.

More important for the performance evaluation of RiTTM is however serendipity. Looking at Table 7.1, a large proportion of the new and relevant test cases in risk areas 3, 4 and 5 were surprising, in the sense that they were not found in the expected test modules. In risk area 4, the original selection contained test cases originating from test modules that did not map to the risk area specification. This indicates that the test engineer was aware that there were relevant test cases in other modules. Without this domain knowledge, the serendipity achieved by RiTTM in risk area 4 would be even higher, indicated in square brackets. From the 29 relevant and new test cases, i. e. test cases that the expert did not choose in his original selection but were then selected by the non-expert with support of RiTTM and then regarded as relevant by the expert, 17 (59%) were serendipitous, i. e. located in unexpected test modules. These results provide evidence for our initial conjecture that topic models can support test engineers to find relevant and unexpected test cases.

## 6 CONCLUSIONS

Since the dawn of humanity we invent and build tools that facilitate progress. While in the past we enhanced our physical capabilities with machines, we strive now to extend our capacity to process information. One, non-invasive, approach to address this challenge is to reduce the amount of information we need to process such that we can focus on gaining new insights, that is, progress.

In this chapter we proposed an approach, RiTTM, for test case selection that does not replace human expertise, but enhances it by providing alternative view to known information. We used topic models to analyze textual data in test cases, reorganizing the structure of an existing test case classification, and providing test engineers the ability to find relevant but unexpected test cases. Our approach encompasses the parameter selection for the topic model, evaluating its intrinsic quality. In a case study performed in collaboration with our industry partner, we evaluated the model's fitness for purpose, i.e. its extrinsic quality. We compared the test case selection performed by an expert test engineer with the results achieved by a non-expert using RiTTM. Using the baseline of inspecting all test cases (1415) and the test cases originally selected by the expert, we found that RiTTM reduces the search space by 79.5–94.7%, allowing the non-expert to achieve a coverage between 15.4–93.7%. We expect that a product domain expert using RiTTM would have achieved a coverage of 98.2%. However, more importantly, RiTTM led to the discovery of 29 relevant test cases, of which the majority (59%) were serendipitous, i.e. not expected.

Although we have developed RiTTM for a specific task at our case company, the design principles, such as model selection and evaluation with respect to topic interpretability, model representation and use, are generic such that solutions for similar challenges in other contexts and companies can be based upon our approach.

### 6.1 Future Work

While research strives at gaining an in-depth and detailed understanding of topic model characteristics, we need also transfer this knowledge into practical applications that support industry in achieving their goals. For RiTTM in particular, but also for the use of topic models in industry in general, we regard the following two avenues for future work as crucial. First, the visualization and interaction with a topic model using a spreadsheet is functional but not intuitive. Recent work on interacting with topic models generated from literary and scientific works [Alexander et al., 2014] shows large potential as it allows a flexible and interactive exploration of

the topic and document space. Second, to establish the adoption of topic models in industry, we need to better understand data aging effects and their costs. There, we see potential in studying topic models that are both interpretable and generalize well on unseen data.

---

## BIBLIOGRAPHY

---

- Aharon Abadi, Mordechai Nisenson, and Yahalomit Simionovici. A Traceability Technique for Specifications. In *Proceedings 16th International Conference on Program Comprehension (ICPC)*, pages 103–112, Amsterdam, The Netherlands, 2008. IEEE. (Cited on pages [34](#), [210](#), [228](#), and [232](#).)
- Fredrik Abbors, Dragos Truscan, and Johan Lilius. Tracing Requirements in a Model-Based Testing Approach. In *Proceedings 1st International Conference on Advances in System Testing and Validation Lifecycle (VALID)*, pages 123–128, Porto, Portugal, 2009. IEEE. (Cited on pages [109](#) and [117](#).)
- Özlem Albayrak, Hülya Kurtoğlu, and Mert Biçakçı. Incomplete Software Requirements and Assumptions Made by Software Engineers. In *Proceedings 16th Asia-Pacific Software Engineering Conference (APSEC)*, pages 333–339, Penang, Malaysia, 2009. IEEE. (Cited on page [1](#).)
- Jonathan Aldrich, Craig Chambers, and David Notkin. ArchJava: connecting software architecture to implementation. In *Proceedings 24th International Conference on Software Engineering (ICSE)*, pages 187–197, Orlando, USA, 2002. IEEE. (Cited on pages [3](#) and [91](#).)
- Eric Alexander, Joe Kohlmann, Robin Valenza, Michael Witmore, and Michael Gleicher. Serendip: Topic model-driven visual exploration of text corpora. In *Proceedings Conference on Visual Analytics Science and Technology (VAST)*, pages 173–182, Paris, France, 2014. IEEE. (Cited on pages [234](#) and [259](#).)
- Ståle Amland. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software*, 53(3):287–295, 2000. (Cited on pages [6](#) and [235](#).)
- Daniel Amyot and Gunter Mussbacher. Bridging the requirements/design gap in dynamic systems with use case maps (UCMs). In *Proceedings 23rd International Conference on Software Engineering (ICSE)*, pages 743–744, Toronto, Canada, 2001. IEEE. (Cited on pages [3](#) and [91](#).)
- Paul André, M. C. Schraefel, Jaime Teevan, and Susan T. Dumais. Discovery is Never by Chance: Designing for (Un)Serendipity. In *Proceedings 7th Conference on Creativity and Cognition*, pages 305–314, Berkeley, USA, 2009. ACM. (Cited on pages [234](#) and [242](#).)
- Giuliano Antoniol, Gerardo Canfora, Andrea De Lucia, and Ettore Merlo. Recovering code to documentation links in OO systems. In *Proceedings*

*6th Working Conference on Reverse Engineering (WCRE)*, pages 136–144, Atlanta, USA, 1999. IEEE. (Cited on pages [6](#) and [196](#).)

Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, and Andrea De Lucia. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings International Conference on Software Maintenance (ICSM)*, pages 40–49, San Jose, USA, 2000. IEEE. (Cited on pages [6](#) and [196](#).)

Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *Transactions on Software Engineering*, 28(10):970 – 983, 2002. (Cited on pages [34](#) and [206](#).)

Andrea Arcuri and Gordon Fraser. On Parameter Tuning in Search Based Software Engineering. In *Proceedings 3rd International Conference on Search Based Software Engineering (SSBSE)*, pages 33–47, Szeged, Hungary, 2011. Springer. (Cited on page [211](#).)

Juan Ares, Rafael García, Natalia Juristo, Marta López, and Ana M. Moreno. A more rigorous and comprehensive approach to software process assessment. *Software Process: Improvement and Practice*, 5(1):3–30, 2000. (Cited on page [26](#).)

Dave Arnold, Jean-Pierre Corriveau, and Wei Shi. Scenario-Based Validation: Beyond the User Requirements Notation. In *Proceedings 21st Australian Software Engineering Conference (ASWEC)*, pages 75–84, Auckland, Australia, 2010. IEEE. (Cited on page [109](#).)

Hazeline U. Asuncion, Arthur U. Asuncion, and Richard N. Taylor. Software Traceability with Topic Modeling. In *Proceedings 32nd International Conference on Software Engineering (ICSE)*, pages 95–104, Cape Town, South Africa, 2010. ACM. (Cited on pages [211](#) and [240](#).)

Iman Avazpour, Teerat Pitakrat, Lars Grunske, and John Grundy. Dimensions and Metrics for Evaluating Recommendation Systems. In *Recommendation Systems in Software Engineering*, pages 245–273. Springer, 2014. (Cited on page [242](#).)

Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004. (Cited on page [94](#).)

Muhammad Ali Babar, Lianping Chen, and Forrest Shull. Managing Variability in Software Product Lines. *IEEE Software*, 27(3):89–91, 94, 2010. (Cited on page [193](#).)

James Bach. Risk and Requirements-Based Testing. *Computer*, 32(6):113–114, 1999. (Cited on page 1.)

Jim Baglama and Lothar Reichel. irlba: Fast partial SVD by implicitly-restarted Lanczos bidiagonalization, January 2014. URL <http://cran.r-project.org/web/packages/irlba/index.html>. (Cited on page 216.)

Zeinab Alizadeh Barmi, Amir Hossein Ebrahimi, and Robert Feldt. Alignment of Requirements Specification and Testing: A Systematic Mapping Study. In *Proceedings 4th International Conference on Software Testing, Verification and Validation (ICST)*, pages 476–485, Berlin, Germany, 2011. IEEE. (Cited on pages 38, 39, 111, and 115.)

Victor R. Basili and Gianluigi Caldiera. Improve software quality by reusing knowledge and experience. *Sloan Management Review*, 37(1):55–64, 1995. (Cited on page 198.)

Martin W. Bauer and George Gaskell. *Qualitative Researching with Text, Image and Sound: A Practical Handbook for Social Research*. SAGE, 2000. (Cited on page 146.)

Kent Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, October 1999. (Cited on pages 2 and 91.)

Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Sutherland Jeff, and Dave Thomas. Manifesto for agile software development, 2001. URL <http://agilemanifesto.org/>. (Cited on pages 4, 33, and 175.)

Andrew Begel and Nachiappan Nagappan. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. In *Proceedings 1st International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 255–264, Madrid, Spain, 2007. IEEE. (Cited on pages 4 and 33.)

Ralf Bender and Stefan Lange. Adjusting for multiple testing - when and how? *Journal of Clinical Epidemiology*, 54(4):343–349, 2001. (Cited on pages 228, 229, and 230.)

Michael W. Berry. SVDPACKC, 2014. URL <http://www.netlib.org/svdpack/>. (Cited on pages 210 and 211.)

Michael W. Berry, Dani Mehzer, Bernard Philippe, and Ahmed Sameh. Parallel Algorithms for the Singular Value Decomposition. In *Handbook of Parallel Computing and Statistics*. CRC Press, 1st edition, 2006. (Cited on page 216.)

- Antonia Bertolino. Software Testing Research: Achievements, Challenges, Dreams. In *Proceedings Workshop on the Future of Software Engineering (FOSE)*, pages 85–103, Minneapolis, USA, 2007. IEEE. (Cited on pages 22, 30, 90, 114, 121, and 137.)
- Lauren R. Biggers, Cecylia Bocovich, Riley Capshaw, Brian P. Eddy, Letha H. Etzkorn, and Nicholas A. Kraft. Configuring latent Dirichlet allocation based feature location. *Empirical Software Engineering*, 19(3):465–500, 2014. (Cited on pages 203 and 229.)
- Ted J. Biggerstaff, Bharat G. Mitbander, and Dallas Webster. The Concept Assignment Problem in Program Understanding. In *Proceedings 15th International Conference on Software Engineering (ICSE)*, pages 482–498, Baltimore, USA, 1993. IEEE. (Cited on pages 6 and 196.)
- David Binkley, Daniel Heinz, Dawn Lawrie, and Justin Overfelt. Understanding LDA in Source Code Analysis. In *Proceedings 22nd International Conference on Program Comprehension (ICPC)*, pages 26–36, Hyderabad, India, 2014. ACM. (Cited on page 243.)
- Andreas Birk, Torgeir Dingsøyr, and Tor Stålhane. Postmortem: Never Leave a Project without It. *IEEE Software*, 19(3):43–45, 2002. (Cited on pages 3, 26, and 138.)
- Elizabeth Bjarnason. *Integrated Requirements Engineering – Understanding and Bridging Gaps in Software Development*. PhD thesis, Lund University, 2013. URL <http://lup.lub.lu.se/record/4117175/file/4117182.pdf>. (Cited on page 23.)
- Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. Requirements are slipping through the gaps - A case study on causes & effects of communication gaps in large-scale software development. In *Proceedings 19th Requirements Engineering Conference (RE)*, pages 37–46, Trento, Italy, 2011. IEEE. (Cited on page 33.)
- Elizabeth Bjarnason, Richard B. Svensson, and Björn Regnell. Evidence-based timelines for project retrospectives - A method for assessing requirements engineering in context. In *Proceedings 2nd International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 17–24, Chicago, USA, 2012. IEEE. (Cited on page 141.)
- Elizabeth Bjarnason, Per Runeson, Markus Borg, Michael Unterkalmsteiner, Emelie Engström, Björn Regnell, Giedre Sabaliauskaite, Annabella Loconsole, Tony Gorschek, and Robert Feldt. Challenges and Practices in Aligning Requirements with Verification and Validation: A Case Study of Six Companies. *Empirical Software Engineering*, 19(6):1809–1855, 2014a. (Cited on pages 138, 139, 176, 185, and 186.)

Elizabeth Bjarnason, Kari Smolander, Emelie Engström, and Per Runeson. Alignment Practices Affect Distances in Software Development: A Theory and a Model. In *Proceedings 3rd SEMAT Workshop on General Theories of Software Engineering*, pages 21–31, Hyderabad, India, 2014b. ACM. (Cited on pages 21, 30, and 36.)

David M. Blei. Probabilistic Topic Models. *Communications of the ACM*, 55(4):77–84, 2012. (Cited on pages 234, 237, and 238.)

David M. Blei and John D. Lafferty. Correlated topic models. In *Proceedings Advances in Neural Information Processing Systems (NIPS)*, pages 147–154, Vancouver, Canada, 2005. (Cited on page 238.)

David M. Blei and John D. Lafferty. Dynamic Topic Models. In *Proceedings 23rd International Conference on Machine Learning (ICML)*, pages 113–120, Pittsburgh, USA, 2006. ACM. (Cited on page 238.)

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003. (Cited on pages 29, 197, 234, 238, 239, 243, and 256.)

Bruce I. Blum. A taxonomy of software development methods. *Communications of the ACM*, 37(11):82–94, 1994. (Cited on page 94.)

Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988. (Cited on pages 2 and 91.)

Barry W. Boehm. Software risk management: principles and practices. *IEEE Software*, 8(1):32–41, 1991. (Cited on page 235.)

Markus Borg. *From Bugs to Decision Support - Leveraging Historical Issue Reports in Software Evolution*. PhD thesis, Lund University, 2015. URL [http://cs.lth.se/fileadmin/cs/Markus\\_Borg/BorgThesis\\_final.pdf](http://cs.lth.se/fileadmin/cs/Markus_Borg/BorgThesis_final.pdf). (Cited on page 23.)

Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014. (Cited on pages 6, 33, 190, 196, 197, 209, 210, and 211.)

Pierre Bourque and Robert Dupuis, editors. *Guide to the Software Engineering Body of Knowledge*. IEEE, 1st edition, 2004. (Cited on page 94.)

Pierre Bourque and Richard E. Fairley, editors. *Guide to the Software Engineering Body of Knowledge*. IEEE, 3rd edition, 2014. (Cited on pages 1 and 137.)

- Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra and its Applications*, 415(1):20–30, 2006. (Cited on page [218](#).)
- Lionel C. Briand, Yvan Labiche, and S. He. Automating regression test selection based on UML designs. *Information and Software Technology*, 51(1):16–30, 2009. (Cited on page [234](#).)
- Samuel Brody and Mirella Lapata. Bayesian Word Sense Induction. In *Proceedings 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 103–111, Athens, Greece, 2009. Association for Computational Linguistics. (Cited on page [237](#).)
- Vanda Broughton. *Essential Classification*. Facet Publishing, 1st edition, 2004. (Cited on page [94](#).)
- Manfred Broy. Challenges in Automotive Software Engineering. In *Proceedings 28th International Conference on Software Engineering (ICSE)*, pages 33–42, Shanghai, China, 2006. ACM. (Cited on page [193](#).)
- Mary Brydon-Miller, Davydd Greenwood, and Patricia Maguire. Why Action Research? *Action Research*, 1(1):9–28, 2003. (Cited on page [11](#).)
- Adolf-Peter Bröhl and Wolfgang Dröschel. *Das V- Modell. Der Standard in der Softwareentwicklung mit Praxisleitfaden*. Oldenbourg Verlag, 1995. (Cited on pages [2](#) and [91](#).)
- Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kriesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005. (Cited on page [94](#).)
- Christian Bunse, Felix C. Freiling, and Nicole Levy. A taxonomy on component-based software engineering methods. In *Architecting Systems with Trustworthy Components*, volume 3938 of *Lecture Notes in Computer Science*, pages 103–119. Springer Verlag, 2006. (Cited on page [94](#).)
- John Calcote. *Autotools: A Practitioner’s Guide to GNU Autoconf, Automake, and Libtool*. No Starch Press, 2010. (Cited on page [205](#).)
- Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, 2008. (Cited on pages [42](#) and [86](#).)
- Marcelo Cataldo, Patrick A. Wagstrom, James D. Herbsleb, and Kathleen M. Carley. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In *Proceedings 20th Anniversary Conference on Computer Supported Cooperative Work*

(CSCW), pages 353–362, Banff, Canada, 2006. ACM. (Cited on pages [5](#) and [140](#).)

Adnan Causevic, Daniel Sundmark, and Sasikumar Punnekkat. Factors Limiting Industrial Adoption of Test Driven Development: A Systematic Review. In *Proceedings 4th International Conference on Software Testing, Verification and Validation (ICST)*, pages 337–346, Berlin, Germany, 2011. IEEE. (Cited on page [177](#).)

Jonathan Chang, Sean Gerrish, Chong Wang, Jordan L. Boyd-Graber, and David M. Blei. Reading Tea Leaves: How Humans Interpret Topic Models. In *Proceedings 23rd Annual Conference on Advances in Neural Information Processing Systems (NIPS)*, pages 288–296, Vancouver, Canada, 2009. Curran Associates, Inc. (Cited on pages [239](#), [241](#), [244](#), [249](#), [250](#), [253](#), and [256](#).)

Thomas Chau, Frank Maurer, and Grigori Melnik. Knowledge Sharing: Agile Methods vs. Tayloristic Methods. In *Proceedings 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 302–307, Linz, Austria, 2003. IEEE. (Cited on page [171](#).)

Lianping Chen and Muhammad Ali Babar. Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges. In *Proceedings 14th International Conference on Software Product Lines: Going Beyond (SPLC)*, pages 166–180, Jeju Island, South Korea, 2010. Springer. (Cited on page [247](#).)

Lianping Chen and Muhammad Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4):344–362, 2011. (Cited on page [193](#).)

Yanping Chen, Robert L. Probert, and D. Paul Sims. Specification-based Regression Test Selection with Risk Analysis. In *Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research*, pages 1–12, Toronto, Canada, 2002. IBM Press. (Cited on pages [234](#) and [236](#).)

Betty H.C. Cheng and Joanne M. Atlee. Research Directions in Requirements Engineering. In *Proceedings Workshop on the Future of Software Engineering (FOSE)*, pages 285–303, Minneapolis, USA, 2007. IEEE. (Cited on pages [22](#), [30](#), [38](#), [90](#), [114](#), and [121](#).)

Brendan Cleary, Chris Exton, Jim Buckley, and Michael English. An empirical analysis of information retrieval based concept location techniques in software comprehension. *Empirical Software Engineering*, 14(1):93–130, 2009. (Cited on pages [34](#), [196](#), [206](#), [210](#), [228](#), and [232](#).)

Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *Transactions on Software Engineering*, 29(9):796–810, 2003. (Cited on pages 41 and 93.)

Jane Cleland-Huang, Adam Czauderna, Alex Dekhtyar, Olly Gotel, Jane Huffman Hayes, Ed Keenan, Greg Leach, Jonathan Maletic, Denys Poshyvanyk, Youghee Shin, Andrea Zisman, Giuliano Antoniol, Brian Berenbach, Alexander Egyed, and Patrick Maeder. Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community. In *Proceedings 6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 17–23, Honolulu, USA, 2011. ACM. (Cited on page 210.)

Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, Boston, 3rd edition edition, 2001. (Cited on page 193.)

David Cohen, Mikael Lindvall, and Patricia Costa. An Introduction to Agile Methods. In *Advances in Computers*, volume 62, pages 1–66. Elsevier, 2004. (Cited on page 171.)

Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional, Boston, 2004. (Cited on page 176.)

Bonnie Collier, Tom DeMarco, and Peter Fearey. A defined process for project post mortem review. *IEEE Software*, 13(4):65–72, 1996. (Cited on pages 26, 138, and 141.)

Mirko Conrad, Ines Fey, and Sadegh Sadeghipour. Systematic Model-Based Testing of Embedded Automotive Software. In *Electronic Notes in Theoretical Computer Science*, volume 111, pages 13–26. Elsevier, January 2005. (Cited on pages 109 and 117.)

Anna Corazza, Sergio Di Martino, and Valerio Maggio. LINSEN: An efficient approach to split identifiers and expand abbreviations. In *Proceedings 28th International Conference on Software Maintenance (ICSM)*, pages 233–242, Trento, Italy, 2012. IEEE. (Cited on page 201.)

Michael J. Crawley. *The R Book*. John Wiley & Sons, 1st edition edition, 2007. (Cited on page 210.)

John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, Inc, Thousand Oaks, 4th edition, 2013. (Cited on pages 10, 11, and 15.)

Jane K. Cullum and Ralph A. Willoughby. *Lanczos Algorithms for Large Symmetric Eigenvalue Computations: Vol. 1: Theory*. SIAM, 2nd edition, 2002. (Cited on page 211.)

Daniela Damian and James Chisan. An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management. *Transactions on Software Engineering*, 32(7):433–453, 2006. (Cited on pages [1](#), [33](#), [38](#), [40](#), and [87](#).)

Daniela Damian, James Chisan, Lakshminarayanan Vaidyanathasamy, and Yogendra Pal. Requirements engineering and downstream software development: Findings from a case study. *Empirical Software Engineering*, 10(3):255–283, 2005. (Cited on pages [39](#), [40](#), [41](#), [87](#), [93](#), [109](#), and [117](#).)

Lars-Ola Damm, Lars Lundberg, and Claes Wohlin. Faults-slip-through—a concept for measuring the efficiency of the test process. *Software Process: Improvement and Practice*, 11(1):47–59, 2006. (Cited on page [233](#).)

Alan M. Davis. *Just enough requirements management: where software development meets marketing*. Dorset House Pub., New York, 2005. (Cited on pages [175](#) and [184](#).)

Alan M. Davis, Scott Overmyer, Kathleen Jordan, Joseph Caruso, Fatma Dandashi, Anhtuan Dinh, Gary Kincaid, Glen Ledeboer, Patricia Reynolds, Pradip Sitaram, Anh Ta, and Mary Theofanos. Identifying and measuring quality in a software requirements specification. In *Proceedings 1st International Software Metrics Symposium (METRICS)*, pages 141–152, Baltimore, USA, May 1993. IEEE. (Cited on page [176](#).)

Robert Davison, Maris G. Martinsons, and Ned Kock. Principles of canonical action research. *Information Systems Journal*, 14(1):65–86, 2004. (Cited on pages [14](#) and [15](#).)

Guido de Caso, Víctor Braberman, Diego Garbervetsky, and Sebastián Uchitel. Automated Abstractions for Contract Validation. *Transactions on Software Engineering*, 38(1):141–162, 2012. (Cited on page [121](#).)

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Can Information Retrieval Techniques Effectively Support Traceability Link Recovery? In *Proceedings 14th International Conference on Program Comprehension (ICPC)*, pages 307–316, Athens, Greece, 2006. IEEE. (Cited on page [196](#).)

Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *Transactions on Software Engineering and Methodology*, 16(4), 2007. (Cited on pages [41](#), [93](#), [196](#), and [211](#).)

Andrea De Lucia, Rocco Oliveto, and Genoveffa Tortora. Assessing IR-based traceability recovery tools through controlled experiments. *Empirical Software Engineering*, 14(1):57–92, 2009. (Cited on page [196](#).)

- Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Improving IR-based Traceability Recovery Using Smoothing Filters. In *Proceedings 19th International Conference on Program Comprehension (ICPC)*, pages 21–30, Kingston, Canada, 2011. IEEE. (Cited on page 196.)
- Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Labeling source code with information retrieval methods: an empirical study. *Empirical Software Engineering*, 19(5):1383–1420, 2013. (Cited on page 240.)
- Valdivino A. de Santiago Júnior and Nandamudi L. Vijaykumar. Generating model-based test cases from natural language requirements for space application software. *Software Quality Journal*, 20(1):77–143, 2012. (Cited on pages 92, 108, 132, and 135.)
- Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990. (Cited on pages 28, 201, 210, 211, 212, 217, and 237.)
- Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A Survey on Model-based Testing Approaches: A Systematic Review. In *Proceedings 1st International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, pages 31–36, Atlanta, USA, 2007. ACM. (Cited on page 41.)
- Torgeir Dingsøyr. Postmortem reviews: purpose and approaches in software engineering. *Information and Software Technology*, 47(5):293–303, 2005. (Cited on pages 26 and 138.)
- Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. Can Better Identifier Splitting Techniques Help Feature Location? In *Proceedings 19th International Conference on Program Comprehension (ICPC)*, pages 11–20, Kingston, Canada, 2011a. IEEE. (Cited on page 201.)
- Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software Maintenance and Evolution: Research and Practice*, 25(1):53–95, 2011b. (Cited on pages 190, 195, 197, 198, 209, 210, and 211.)
- Bogdan Dit, Annibale Panichella, Evan Moritz, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. Configuring topic models for software engineering tasks in TraceLab. In *Proceedings 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 105–109, San Francisco, USA, 2013. IEEE. (Cited on page 210.)

Bogdan Dit, Evan Moritz, Mario Linares-Vásquez, Denys Poshyvanyk, and Jane Cleland-Huang. Supporting and accelerating reproducible empirical research in software evolution and maintenance using TraceLab Component Library. *Empirical Software Engineering*, pages 1–39, 2014. (Cited on page 210.)

R. Geoff Dromey. From requirements to design: formalizing the key steps. In *Proceedings 1st International Conference on Software Engineering and Formal Methods (SEFM)*, pages 2–11, Brisbane, Australia, 2003. IEEE. (Cited on page 176.)

Susan T. Dumais. LSI meets TREC: a status report. In *NIST special publication*, pages 137–152. National Institute of Standards and Technology, 1992. (Cited on page 211.)

Tore Dybå. An empirical investigation of the key factors for success in software process improvement. *Transactions on Software Engineering*, 31(5):410–424, 2005. (Cited on page 145.)

Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10):833–859, 2008. (Cited on pages 4 and 33.)

Tore Dybå, Barbara A. Kitchenham, and Magne Jørgensen. Evidence-based software engineering for practitioners. *IEEE Software*, 22(1):58–65, 2005. (Cited on page 198.)

Ralf Dömges and Klaus Pohl. Adapting Traceability Environments to Project-specific Needs. *Communications of the ACM*, 41(12):54–62, 1998. (Cited on page 32.)

Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer, London, UK, 2008. (Cited on pages 10, 11, 12, 13, 14, 15, and 23.)

Christof Ebert and Capers Jones. Embedded Software: Facts, Figures, and Future. *Computer*, 42(4):42–52, 2009. (Cited on page 193.)

Amnon H. Eden. Three Paradigms of Computer Science. *Minds and Machines*, 17(2):135–167, 2007. (Cited on page 11.)

Eugene Edgington and Patrick Onghena. *Randomization Tests*. Chapman and Hall/CRC, Boca Raton, FL, 4th edition edition, 2007. (Cited on pages 223 and 230.)

- Mohamed El-Attar and James Miller. Developing comprehensive acceptance tests from use cases and robustness diagrams. *Requirements Engineering*, 15(3):285–306, 2010. (Cited on pages [108](#), [115](#), [117](#), [132](#), and [136](#).)
- Frank Elberzhager, Alla Rosbach, Jürgen Münch, and Robert Eschbach. Reducing test effort: A systematic mapping study on existing approaches. *Information and Software Technology*, 54(10):1092–1106, 2012. (Cited on page [235](#).)
- Tzilla Elrad, Omar Aldawud, and Atef Bader. Aspect-Oriented Modeling: Bridging the Gap between Implementation and Design. In *Proceedings 1st Conference on Generative Programming and Component Engineering (GPCE)*, pages 189–201, Pittsburgh, USA, 2002. Springer. (Cited on pages [3](#) and [91](#).)
- Mica R. Endsley and David B. Kaber. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42(3):462–492, 1999. (Cited on page [240](#).)
- Emelie Engström and Per Runeson. Software product line testing - A systematic mapping study. *Information and Software Technology*, 53(1):2–13, 2011. (Cited on page [193](#).)
- Emelie Engström, Per Runeson, and Mats Skoglund. A systematic review on regression test selection techniques. *Information and Software Technology*, 52(1):14–30, 2010. (Cited on pages [5](#), [193](#), and [236](#).)
- Eric Enslen, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *Proceedings 6th International Working Conference on Mining Software Repositories (MSR)*, pages 71–80, Vancouver, Canada, 2009. IEEE. (Cited on page [201](#).)
- Davide Falessi, Giovanni Cantone, and Gerardo Canfora. Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques. *Transactions on Software Engineering*, 39(1):18–44, 2013. (Cited on pages [32](#), [195](#), [197](#), [200](#), [201](#), [203](#), [207](#), [227](#), [229](#), [230](#), [231](#), [232](#), and [249](#).)
- Li Fei-Fei and Pietro Perona. A Bayesian hierarchical model for learning natural scene categories. In *Proceedings International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 524–531, San Diego, USA, 2005. IEEE. (Cited on page [237](#).)
- Ingo Feinerer, Kurt Hornik, and David Meyer. Text Mining Infrastructure in R. *Journal of Statistical Software*, 25(5):1–54, 2008. (Cited on page [213](#).)

Michael Felderer and Rudolf Ramler. A multiple case study on risk-based testing in industry. *International Journal on Software Tools for Technology Transfer*, 16(5):609–625, 2014. (Cited on page 235.)

Michael Felderer and Ina Schieferdecker. A taxonomy of risk-based testing. *International Journal on Software Tools for Technology Transfer*, 16(5):559–568, 2014. (Cited on pages 3, 235, 236, and 244.)

Stuart I. Feldman. Make - a program for maintaining computer programs. *Software: Practice and Experience*, 9(4):255–265, 1979. (Cited on page 205.)

Robert Feldt. *Biomimetic Software Engineering Techniques for Dependability*. PhD thesis, Department of Computer Engineering, Chalmers University of Technology, Gothenburg, Sweden, December 2002. URL [http://www.cse.chalmers.se/~feldt/publications/feldt\\_2002\\_phdthesis.html](http://www.cse.chalmers.se/~feldt/publications/feldt_2002_phdthesis.html). (Cited on page 99.)

Robert Feldt. Do System Test Cases Grow Old? In *Proceedings 7th International Conference on Software Testing, Verification and Validation (ICST)*, pages 343–352, Cleveland, USA, 2014. IEEE. (Cited on pages 234 and 237.)

Robert W. Ferguson and Giuseppe Lami. An Empirical Study on the Relationship between Defective Requirements and Test Failures. In *Proceedings 30th Annual Software Engineering Workshop (SEW)*, pages 7–10, Columbia, USA, 2006. IEEE. (Cited on page 84.)

Francesco Flammini, Nicola Mazzocca, and Antonio Orazzo. Automatic instantiation of abstract tests on specific configurations for large critical control systems. *Software Testing, Verification and Reliability*, 19(2):91–110, 2009. (Cited on pages 92, 108, 115, 132, and 134.)

Luciano Floridi. *Information: A Very Short Introduction*. Oxford University Press, Usa, 1st edition, 2010. (Cited on page 102.)

Nina D. Fogelström and Tony Gorschek. Test-case Driven versus Checklist-based Inspections of Software Requirements – An Experimental Evaluation. In *Proceedings 10th Workshop on Requirements Engineering (WER)*, Toronto, Canada, 2007. (Cited on page 83.)

Kevin Forsberg and Harold Mooz. The Relationship of System Engineering to the Project Cycle. In *Proceedings of the National Council for System Engineering (NCOSE) Conference*, pages 57–65, Chattanooga, USA, 1991. Center for Systems Management. (Cited on pages 2 and 91.)

Martin Fowler and Jim Highsmith. The agile manifesto. *Software Development*, 9(8):28–35, 2001. (Cited on page 171.)

Samuel Fricker, Tony Gorschek, Carl Byman, and Armin Schmidle. Handshaking with Implementation Proposals: Negotiating Requirements Understanding. *IEEE Software*, 27(2):72–80, 2010. (Cited on pages [123](#) and [165](#).)

Tor Erlend Fægri, Tore Dybå, and Torgeir Dingsøyr. Introducing knowledge redundancy practice in software development: Experiences with job rotation in support work. *Information and Software Technology*, 52(10): 1118–1132, 2010. (Cited on page [4](#).)

Laura V. Galvis Carreño and Kristina Winbladh. Analysis of User Comments: An Approach for Software Requirements Evolution. In *Proceedings 35th International Conference on Software Engineering (ICSE)*, pages 582–591, San Francisco, USA, 2013. IEEE. (Cited on page [240](#).)

Florent Garcin and Boi Faltings. PEN Recsys: A Personalized News Recommender Systems Framework. In *Proceedings 7th International News Recommender Systems Workshop and Challenge (RecSys)*, pages 3–9, Hong Kong, China, 2013. ACM. (Cited on page [234](#).)

Gregory Gay, Sonia Haiduc, Adrian Marcus, and Tim Menzies. On the use of relevance feedback in IR-based concept location. In *Proceedings 28th International Conference on Software Maintenance (ICSM)*, pages 351–360, Edmonton, Canada, 2009. IEEE. (Cited on pages [210](#) and [216](#).)

Boby George and Laurie Williams. A structured experiment of test-driven development. *Information and Software Technology*, 46(5):337–342, 2004. (Cited on page [177](#).)

Malcolm Gethers and Denys Poshyvanyk. Using Relational Topic Models to capture coupling among classes in object-oriented software systems. In *Proceedings 26th International Conference on Software Maintenance (ICSM)*, pages 1–10, Timisoara, Romania, 2010. IEEE. (Cited on page [240](#).)

Malcolm Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proceedings 27th International Conference on Software Maintenance (ICSM)*, pages 133–142, Williamsburg, USA, 2011. IEEE. (Cited on pages [32](#) and [196](#).)

Barney Glaser, Anselm Strauss, and Elizabeth Strutzel. The discovery of grounded theory: strategies for qualitative research. *Nursing Research*, 17 (4), 1968. (Cited on page [10](#).)

Robert L. Glass. Project retrospectives, and why they never happen. *IEEE Software*, 19(5):112–111, 2002a. (Cited on pages [26](#), [138](#), [141](#), and [173](#).)

Robert L. Glass. Sorting out software complexity. *Communications of the ACM*, 45(11):19–21, 2002b. (Cited on pages [1](#) and [92](#).)

Robert L. Glass and Iris Vessey. Contemporary Application-Domain Taxonomies. *IEEE Software*, 12(4):63–76, 1995. (Cited on pages [93](#), [95](#), and [96](#).)

Robert L. Glass, Iris Vessey, and Venkataraman Ramesh. Research in software engineering: an analysis of the literature. *Information and Software Technology*, 44(8):491–506, 2002. (Cited on page [94](#).)

David E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, Boston, USA, 1st edition, 1989. (Cited on page [197](#).)

Tony Gorschek and Alan M. Davis. Requirements engineering: In search of the dependent variables. *Information and Software Technology*, 50(1-2): 67–75, 2008. (Cited on pages [38](#), [40](#), [49](#), [123](#), and [124](#).)

Tony Gorschek and Claes Wohlin. Requirements abstraction model. *Requirements Engineering*, 11(1):79–101, 2006. (Cited on pages [3](#), [49](#), and [108](#).)

Tony Gorschek, Claes Wohlin, Per Carre, and Stig Larsson. A Model for Technology Transfer in Practice. *IEEE Software*, 23(6):88–95, 2006. (Cited on pages [21](#), [22](#), [142](#), and [190](#).)

Tony Gorschek, Ewan Tempero, and Lefteris Angelis. On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software*, 95:176–193, 2014. (Cited on page [234](#).)

Orlena Gotel and Anthony C. W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings 1st International Conference on Requirements Engineering (RE)*, pages 94–101, Colorado Springs, USA, 1994. IEEE. (Cited on pages [6](#), [41](#), [92](#), [93](#), [117](#), and [196](#).)

Orlena Gotel, Jane Cleland-Huang, Jane Huffman Hayes, Andrea Zisman, Alexander Egyed, Paul Grunbacher, and Giuliano Antoniol. The quest for Ubiquity: A roadmap for software and systems traceability research. In *Proceedings 20th Requirements Engineering Conference (RE)*, pages 71–80, Chicago, USA, 2012. IEEE. (Cited on pages [6](#) and [32](#).)

Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: the state of the practice. *IEEE Software*, 20(6):61–69, 2003. (Cited on pages [191](#) and [247](#).)

- Dorothy Graham. Requirements and testing: seven missing-link myths. *IEEE Software*, 19(5):15–17, 2002. (Cited on pages [3](#), [22](#), [30](#), [84](#), [90](#), [92](#), and [138](#).)
- Scott Grant, James R. Cordy, and David B. Skillicorn. Using heuristics to estimate an appropriate number of latent topics in source code analysis. *Science of Computer Programming*, 78(9):1663–1678, 2013. (Cited on page [231](#).)
- Wolfgang Grieskamp, Nicolas Kicillof, Keith Stobie, and Victor Braberman. Model-based quality assurance of protocol documentation: tools and methodology. *Software Testing, Verification and Reliability*, 21(1):55–71, 2011. (Cited on pages [42](#) and [121](#).)
- Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101(suppl 1):5228–5235, 2004. (Cited on pages [243](#) and [244](#).)
- Thomas L. Griffiths, Michael I. Jordan, Joshua B. Tenenbaum, and David M. Blei. Hierarchical Topic Models and the Nested Chinese Restaurant Process. In *Proceedings Advances in Neural Information Processing Systems (NIPS)*, pages 17–24, Vancouver and Whistler, Canada, 2003. MIT Press. (Cited on page [238](#).)
- David Grossman and Ophir Frieder. *Information Retrieval - Algorithms and Heuristics*, volume 15 of *The Information Retrieval Series*. Springer, New York, USA, 2nd edition, 2004. (Cited on pages [3](#), [190](#), [195](#), and [200](#).)
- Lars Grunske. Specification patterns for probabilistic quality properties. In *Proceedings 30th International Conference on Software Engineering (ICSE)*, pages 31–40, Leipzig, Germany, 2008. ACM. (Cited on page [111](#).)
- Latifa Guerrouj, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. TIDIER: an identifier splitting approach using speech recognition techniques. *Journal of Software Maintenance and Evolution: Research and Practice*, 25(6):575–599, 2011. (Cited on page [201](#).)
- Baris Güldali, Holger Funke, Stefan Sauer, and Gregor Engels. TORC: test plan optimization by requirements clustering. *Software Quality Journal*, 19(4):771–799, June 2011. (Cited on pages [24](#), [108](#), [132](#), and [133](#).)
- Aria Haghghi and Lucy Vanderwende. Exploring Content Models for Multi-document Summarization. In *Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 362–370, Boulder, USA, 2009. Association for Computational Linguistics. (Cited on page [237](#).)

Jon G. Hall, Michael Jackson, Robin C. Laney, Bashar Nuseibeh, and Luca Rapanotti. Relating software requirements and architectures using problem frames. In *Proceedings 10th International Conference on Requirements Engineering (RE)*, pages 137– 144, Essen, Germany, 2002. IEEE. (Cited on pages [3](#) and [91](#).)

Bill Hasling, Helmut Goetz, and Klaus Beetz. Model Based Testing of System Requirements using UML Use Case Models. In *Proceedings 1st International Conference on Software Testing, Verification, and Validation (ICST)*, pages 367–376, Lillehammer, Norway, 2008. IEEE. (Cited on pages [42](#) and [86](#).)

Børge Haugset and Geir K. Hanssen. Automated Acceptance Testing: A Literature Review and an Industrial Case Study. In *Proceedings Agile 2008 Conference*, pages 27–38, Toronto, Canada, 2008. IEEE. (Cited on pages [177](#), [183](#), [184](#), and [185](#).)

Jane Hayes, Alex Dekhtyar, and Senthil K. Sundaram. Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *Transactions on Software Engineering*, 32(1):4–19, 2006. (Cited on pages [12](#), [19](#), and [111](#).)

Jane Hayes, Alex Dekhtyar, Senthil Sundaram, E. Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007. (Cited on pages [41](#) and [93](#).)

Matthias Heindl and Stefan Biffl. A Case Study on Value-based Requirements Tracing. In *Proceedings 10th European Software Engineering Conference Held Jointly with 13th International Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 60–69, Lisbon, Portugal, 2005. ACM. (Cited on pages [6](#) and [32](#).)

Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated Consistency Checking of Requirements Specifications. *Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996. (Cited on page [176](#).)

Hadi Hemmati, Zhihan Fang, and Mika V. Mäntylä. Prioritizing Manual Test Cases in Traditional and Rapid Release Environments. In *Proceedings 8th International Conference on Software Testing, Verification and Validation (ICST)*, Graz, Austria, 2015. IEEE. (Cited on page [240](#).)

James D. Herbsleb and Rebecca E. Grinter. Splitting the Organization and Integrating the Code: Conway’s Law Revisited. In *Proceedings 21st International Conference on Software Engineering (ICSE)*, pages 85–95, Los Angeles, USA, 1999. ACM. (Cited on pages [5](#) and [140](#).)

- James D. Herbsleb and Audris Mockus. Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering. In *Proceedings 9th European Software Engineering Conference Held Jointly with 11th International Symposium on Foundations of Software Engineering (ESEC/FSE)*, pages 138–137, Helsinki, Finland, 2003. ACM. (Cited on pages 5, 36, and 140.)
- Vicente Hernandez, Jose E. Roman, and Vicente Vidal. SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems. *Transactions on Mathematical Software*, 31(3):351–362, 2005. (Cited on page 218.)
- Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004. (Cited on pages 16 and 142.)
- Emily Hill, Zachary P. Fry, Haley Boyd, Giriprasad Sridhara, Yana Novikova, Lori Pollock, and K. Vijay-Shanker. AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings 5th International Working Conference on Mining Software Repositories (MSR)*, pages 79–88, Leipzig, Germany, 2008. ACM. (Cited on page 201.)
- Abram Hindle, Christian Bird, Thomas Zimmermann, and Nachiappan Nagappan. Do topics make sense to managers and developers? *Empirical Software Engineering*, pages 1–37, 2014. (Cited on pages 239 and 241.)
- Thomas Hofmann. Probabilistic Latent Semantic Indexing. In *Proceedings 22nd Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 50–57, Berkeley, USA, 1999. ACM. (Cited on pages 237 and 243.)
- John N. Hooker. Testing heuristics: We have it all wrong. *Journal of Heuristics*, 1(1):33–42, 1995. (Cited on pages 210 and 211.)
- Siv H. Houmb, Shareeful Islam, Eric Knauss, Jan Jürjens, and Kurt Schneider. Eliciting security requirements and tracing them to design: An integration of Common Criteria, heuristics, and UMLsec. *Requirements Engineering*, 15(1):63–93, 2010. (Cited on page 120.)
- Pei Hsia, David Kung, and Chris Sell. Software requirements and acceptance testing. *Annals of Software Engineering*, 3(1):291–317, 1997. (Cited on page 176.)
- Martin Höst, Robert Feldt, and Frank Luders. Support for Different Roles in Software Engineering Master’s Thesis Projects. *Transactions on Education*, 53(2):288–296, 2010. (Cited on page 49.)

IEEE. IEEE Standard Glossary of Software Engineering Terminology, 1990. (Cited on pages [70](#), [71](#), and [93](#).)

Juhani Iivari and John Venable. Action research and design science research - Seemingly similar but decisively dissimilar. In *Proceedings 17th European Conference on Information Systems (ECIS)*, Verona, Italy, 2009. AIS Electronic Library. (Cited on page [16](#).)

Md. Mahfuzul Islam, Alessandro Marchetto, Angelo Susi, and Giuseppe Scanniello. A Multi-Objective Technique to Prioritize Test Cases Based on Latent Semantic Indexing. In *Proceedings 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 21–30, Szeged, Hungary, 2012. IEEE. (Cited on page [197](#).)

ISO. Software and systems engineering - Software testing - Part 2: Test processes (ISO/IEC/IEEE 29119), 2013. (Cited on page [235](#).)

ISO/IEC. 9126-1:2001(E), International standard software engineering product quality part 1: Quality model, 2001. (Cited on page [70](#).)

ISO/IEC. ISO/IEC Standard for Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems, 2007. (Cited on pages [1](#) and [139](#).)

Martin Ivarsson and Tony Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, 2011. (Cited on pages [15](#), [97](#), and [190](#).)

Martin Ivarsson and Tony Gorschek. Tool support for disseminating and improving development practices. *Software Quality Journal*, 20(1):173–199, 2012. (Cited on pages [26](#) and [138](#).)

Kenneth E. Iverson. Notation As a Tool of Thought. *Communications of the ACM*, 23(8):444–465, 1980. (Cited on page [213](#).)

David S. Janzen and Hossein Saiedian. A Leveled Examination of Test-Driven Development Acceptance. In *Proceedings 29th International Conference on Software Engineering (ICSE)*, pages 719–722, Minneapolis, USA, 2007. IEEE. (Cited on page [177](#).)

Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 1st edition, 2011. (Cited on page [243](#).)

Matthias Jarke. Requirements Tracing. *Communications of the ACM*, 41(12): 32–36, 1998. (Cited on page [41](#).)

Hsin-Yi Jiang, Tien N. Nguyen, Ing-Xiang Chen, Hojun Jaygarl, and Carl K. Chang. Incremental Latent Semantic Indexing for Automatic Traceability Link Evolution Management. In *Proceedings 23rd International Conference on Automated Software Engineering (ASE)*, pages 59–68, L’Aquila, Italy, 2008. IEEE. (Cited on page 218.)

James A. Jones, Mark Grechanik, and Andre van der Hoek. Enabling and Enhancing Collaborations Between Software Development Organizations and Independent Test Agencies. In *Proceedings Workshop on Cooperative and Human Aspects on Software Engineering (CHASE)*, pages 56–59, Vancouver, Canada, 2009. IEEE. (Cited on pages 38 and 83.)

Anne Keegan and J. Rodney Turner. Quantity versus Quality in Project-Based Learning Practices. *Management Learning*, 32(1):77–98, 2001. (Cited on pages 26, 138, and 173.)

Mahvish Khurum, Kai Petersen, and Tony Gorschek. Extending value stream mapping through waste definition beyond customer perspective. *Journal of Software: Evolution and Process*, 26(12):1074–1105, 2014. (Cited on page 171.)

Jung-Min Kim and Adam Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Proceedings 24th International Conference on Software Engineering (ICSE)*, pages 119–129, Orlando, USA, 2002. IEEE. (Cited on page 237.)

Barbara A. Kitchenham and Stuart Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, Software Engineering Group, Keele University and Department of Computer Science, University of Durham, United Kingdom, 2007. (Cited on pages 113 and 115.)

Barbara A. Kitchenham, Tore Dybå, and Magne Jørgensen. Evidence-Based Software Engineering. In *Proceedings 26th International Conference on Software Engineering (ICSE)*, pages 273–281, Edinburgh, UK, 2004. IEEE. (Cited on page 11.)

Jochen Knaus. snowfall: Easier cluster computing (based on snow), December 2013. URL <http://cran.r-project.org/web/packages/snowfall/index.html>. (Cited on page 213.)

Andrew J. Ko, Robert DeLine, and Gina Venolia. Information Needs in Collocated Software Development Teams. In *Proceedings 29th International Conference on Software Engineering (ICSE)*, pages 344–353, Minneapolis, USA, 2007. IEEE. (Cited on page 140.)

Vidar Kongsli. Towards Agile Security in Web Applications. In *Proceedings 21st Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 805–808, Portland, USA, 2006. ACM. (Cited on page [184](#).)

Christian Kop and Heinrich C. Mayr. Conceptual predesign bridging the gap between requirements and conceptual design. In *Proceedings 3rd International Conference on Requirements Engineering (RE)*, pages 90–98, Colorado Springs, USA, 1998. IEEE. (Cited on pages [3](#) and [91](#).)

Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley, 1st edition, 1998. (Cited on page [1](#).)

Robert E. Kraut and Lynn A. Streeter. Coordination in Software Development. *Communications of the ACM*, 38(3):69–81, 1995. (Cited on pages [4](#), [38](#), [87](#), and [139](#).)

Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing, Boston, USA, 2nd edition, 2000. (Cited on pages [2](#) and [91](#).)

Johanna Kukkanen, Kyösti Väkeväinen, Marjo Kauppinen, and Eero Uusitalo. Applying a Systematic Approach to Link Requirements and Testing: A Case Study. In *Proceedings 6th Asia-Pacific Software Engineering Conference (APSEC)*, pages 482–488, Penang, Malaysia, 2009. IEEE. (Cited on pages [30](#), [33](#), [38](#), [40](#), [41](#), [86](#), [87](#), [93](#), and [112](#).)

Debasish Kundu, Monalisa Sarma, Debasis Samanta, and Rajib Mall. System testing for object-oriented systems with test case prioritization. *Software Testing, Verification and Reliability*, 19(4):297–333, 2009. (Cited on page [121](#).)

Barbara H. Kwasnik. The Role of Classification Structures in Reflecting and Building Theory. *Advances in Classification Research Online*, 3(1):63–82, 1992. (Cited on pages [93](#) and [94](#).)

Barbara H. Kwasnik. The role of classification in knowledge representation and discovery. *Library trends*, 48(1):22–47, 1999. (Cited on page [93](#).)

W. Lam and V. Shankararaman. Requirements change: a dissection of management issues. In *Proceedings 25th EUROMICRO Conference*, pages 244–251, Milan, Italy, 1999. IEEE. (Cited on page [1](#).)

Philip A. Laplante. *What Every Engineer Should Know about Software Engineering*. CRC Press, 1st edition, 2007. (Cited on pages [2](#) and [91](#).)

Roberto Latorre. A successful application of a Test-Driven Development strategy in the industrial environment. *Empirical Software Engineering*, 19(3):753–773, 2014. (Cited on pages [183](#) and [184](#).)

- Jey Han Lau, David Newman, and Timothy Baldwin. Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. In *Proceedings 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Gothenburg, Sweden, 2014. Association for Computational Linguistics. (Cited on pages 239, 240, 244, 249, 250, and 253.)
- Soren Lauesen. *Software Requirements: Styles & Techniques*. Addison-Wesley Professional, Harlow, 1st edition, 2002. (Cited on pages 183 and 186.)
- Niklas Lavesson and Paul Davidsson. Quantifying the Impact of Learning Algorithm Parameter Tuning. In *Proceedings 21st National Conference on Artificial Intelligence (AAAI)*, pages 395–400, Boston, USA, 2006. AAAI Press. (Cited on pages 211 and 243.)
- Michael A. Lawrence. ez: Easy analysis and visualization of factorial experiments, 2013. URL <http://cran.r-project.org/web/packages/ez/index.html>. (Cited on page 223.)
- Dawn Lawrie and Dave Binkley. Expanding identifiers to normalize source code vocabulary. In *Proceedings 27th IEEE International Conference on Software Maintenance (ICSM)*, pages 113–122, Williamsburg, USA, 2011. IEEE. (Cited on page 201.)
- Lucas Layman, Laurie Williams, and Lynn Cunningham. Motivations and measurements in an agile case study. *Journal of Systems Architecture*, 52(11):654–667, 2006. (Cited on page 175.)
- Jihyun Lee, Sungwon Kang, and Danhyung Lee. A Survey on Software Product Line Testing. In *Proceedings 16th International Software Product Line Conference (SPLC)*, pages 31–40, Salvador, Brazil, 2012. ACM. (Cited on page 193.)
- Timothy C. Lethbridge, Janice Singer, and Andrew Forward. How software engineers use documentation: the state of the practice. *IEEE Software*, 20(6):35–39, 2003. (Cited on page 177.)
- Paul L. Li, James D. Herbsleb, Mary Shaw, and Brian Robinson. Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc. In *Proceedings 28th International Conference on Software Engineering (ICSE)*, pages 413–422, Shanghai, China, 2006. ACM. (Cited on page 235.)
- Caroli Linnaei. *Systema Naturae: Sive Regna Tria Naturae Systematice Proposita Per Classes, Ordines, Genera, & Species*. Lugduni Batavorum, 1735. (Cited on pages 3 and 93.)

Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. In *Proceedings 22nd International Conference on Automated Software Engineering (ASE)*, pages 234–243, Atlanta, USA, 2007. ACM. (Cited on page 216.)

Shaoying Liu. Integrating top-down and scenario-based methods for constructing software specifications. *Information and Software Technology*, 51(11):1565–1572, 2009. (Cited on page 115.)

Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 1st edition edition, 2011. (Cited on page 226.)

Sugandha Lohar, Sorawit Amornborvornwong, Andrea Zisman, and Jane Cleland-Huang. Improving Trace Accuracy Through Data-driven Configuration and Composition of Tracing Features. In *Proceedings 9th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 378–388, Saint Petersburg, Russia, 2013. ACM. (Cited on pages 36 and 197.)

Marco Lormans and Arie van Deursen. Can LSI help reconstructing requirements traceability in design and test? In *Proceedings 10th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 47–56, Bari, Italy, 2006. IEEE. (Cited on page 210.)

Marco Lormans, Arie Deursen, and Hans-Gerhard Gross. An industrial case study in reconstructing requirements views. *Empirical Software Engineering*, 13(6):727–760, 2008. (Cited on page 41.)

Mitch Lubars, Colin Potts, and Charles Richter. A review of the state of the practice in requirements modeling. In *Proceedings 1st International Symposium on Requirements Engineering*, pages 2–14, San Diego, USA, 1993. IEEE. (Cited on page 42.)

John Ludbrook and Hugh Dudley. Why Permutation Tests are Superior to t and F Tests in Biomedical Research. *The American Statistician*, 52(2):127–132, 1998. (Cited on pages 223 and 230.)

Nazim H. Madhavji. The process cycle [software engineering]. *Software Engineering Journal*, 6(5):234–242, 1991. (Cited on page 89.)

Jonathan I. Maletic and Adrian Marcus. Using latent semantic analysis to identify similarities in source code to support program understanding. In *Proceedings 12th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 46–53, Vancouver, Canada, 2000. IEEE. (Cited on pages 6 and 196.)

Jonathan I. Maletic and Andrian Marcus. Supporting Program Comprehension Using Semantic and Structural Information. In *Proceedings 23rd*

*International Conference on Software Engineering (ICSE)*, pages 103–112, Toronto, Canada, 2001. IEEE. (Cited on pages 6 and 196.)

Jonathan I. Maletic and Naveen Valluri. Automatic software clustering via Latent Semantic Analysis. In *Proceedings 14th International Conference on Automated Software Engineering (ASE)*, pages 251–254, Cocoa Beach, USA, 1999. IEEE. (Cited on pages 6 and 196.)

Jonathan I. Maletic, Michael L. Collard, and Adrian Marcus. Source code files as structured documents. In *Proceedings 10th International Workshop on Program Comprehension (IWPC)*, pages 289–292, Paris, France, 2002. IEEE. (Cited on page 213.)

Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995. (Cited on page 16.)

Adrian Marcus and Jonathan I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings 25th International Conference on Software Engineering (ICSE)*, pages 125 – 135, Portland, USA, 2003. IEEE. (Cited on pages 196 and 206.)

Adrian Marcus, Andrey Sergeyev, Václav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings 11th Working Conference on Reverse Engineering (WCORE)*, pages 214–223, Delft, The Netherlands, 2004. IEEE. (Cited on pages 196 and 206.)

Sabrina Marczak and Daniela Damian. How interaction between roles shapes the communication structure in requirements-driven collaboration. In *Proceedings 19th International Conference on Requirements Engineering (RE)*, pages 47–56, Trento, Italy, 2011. IEEE. (Cited on pages 4 and 92.)

Peter Markie. Rationalism vs. Empiricism. In *The Stanford Encyclopedia of Philosophy*. Spring 2015 edition, 2015. URL <http://plato.stanford.edu/archives/spr2015/entries/rationalism-empiricism/>. (Cited on pages 11 and 18.)

Robert C. Martin and Grigori Melnik. Tests and Requirements, Requirements and Tests: A Möbius Strip. *IEEE Software*, 25(1):54–59, 2008. (Cited on pages 22, 30, 42, 84, 92, 176, and 177.)

Girish Maskeri, Santonu Sarkar, and Kenneth Heafield. Mining Business Topics in Source Code Using Latent Dirichlet Allocation. In *Proceedings 1st India Software Engineering Conference*, pages 113–120, Bangalore, India, 2008. ACM. (Cited on page 240.)

Alistair Mavin and Philip Wilkinson. Big Ears (The Return of "Easy Approach to Requirements Engineering"). In *Proceedings 18th Requirements Engineering Conference (RE)*, pages 277–282, Sydney, Australia, 2010. IEEE. (Cited on page 176.)

Andrew Kachites McCallum. MALLET: A Machine Learning for Language Toolkit, 2002. URL <http://mallet.cs.umass.edu>. (Cited on page 249.)

Sharon McGee and Des Greer. Towards an understanding of the causes and effects of software requirements change: two case studies. *Requirements Engineering*, 17(2):133–155, 2012. (Cited on page 1.)

Colin McMillan, Denys Poshyvanyk, and Meghan Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *Proceedings 5th Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 41–48, Vancouver, Canada, 2009. IEEE. (Cited on page 32.)

Nikunj R. Mehta, Nenad Medvidovic, and Sandeep Phadke. Towards a Taxonomy of Software Connectors. In *Proceedings 22nd International Conference on Software Engineering (ICSE)*, pages 178–187, Limerick, Ireland, 2000. ACM. (Cited on page 94.)

Vineet Mehta, Rajmonda S. Caceres, and Kevin M. Carter. Evaluating topic quality using model clustering. In *Proceedings Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 178–185, Orlando, USA, 2014. IEEE. (Cited on page 244.)

Grigori Melnik and Frank Maurer. Multiple Perspectives on Executable Acceptance Test-driven Development. In *Proceedings 8th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP)*, pages 245–249, Como, Italy, 2007. Springer. (Cited on page 183.)

Grigori Melnik, Frank Maurer, and Mike Chiasson. Executable acceptance tests for communicating business requirements: customer perspective. In *Proceedings Agile 2006 Conference*, pages 12–46, Minneapolis, USA, 2006. IEEE. (Cited on pages 42, 92, and 177.)

Tim Menzies and Thomas Zimmermann. Software Analytics: So What? *IEEE Software*, 30(4):31–37, 2013. (Cited on page 241.)

Donna M. Mertens. Transformative Paradigm. In Sandra Mathison, editor, *Encyclopedia of Evaluation*, pages 423–424. Sage Publications, Inc., 2005. (Cited on page 11.)

Florian Merz, Carsten Sinz, Hendrik Post, Thomas Gorges, and Thomas Kropf. Bridging the gap between test cases and requirements by abstract

testing. *Innovations in Systems and Software Engineering*, pages 1–10, 2015. (Cited on page 22.)

Jani Metsa, Mika Katara, and Tommi Mikkonen. Testing Non-Functional Requirements with Aspects: An Industrial Case Study. In *Proceedings 7th International Conference on Quality Software (QSIC)*, pages 5–14, Portland, USA, 2007. IEEE. (Cited on page 109.)

Tim Miller and Paul Strooper. A case study in model-based testing of specifications and implementations. *Software Testing, Verification and Reliability*, 22(1):33–63, 2010. (Cited on pages 92, 96, 98, 104, 107, 108, 117, and 176.)

David Mimno, Hanna M. Wallach, Edmund Talley, Miriam Leenders, and Andrew McCallum. Optimizing Semantic Coherence in Topic Models. In *Proceedings Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 262–272, Edinburgh, UK, 2011. Association for Computational Linguistics. (Cited on pages 239, 240, 244, and 253.)

Deepti Mishra, Alok Mishra, and Sofiya Ostrovska. Impact of physical ambiance on communication, collaboration and coordination in agile software development: An empirical evaluation. *Information and Software Technology*, 54(10):1067–1078, 2012. (Cited on pages 4 and 33.)

Nikolaos Mittas and Lefteris Angelis. Comparing cost prediction models by resampling techniques. *Journal of Systems and Software*, 81(5):616–632, May 2008. (Cited on page 230.)

Parastoo Mohagheghi and Vegard Dehlen. Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In *Proceedings 4th European Conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA)*, pages 432–443, Berlin, Germany, 2008. Springer. (Cited on page 42.)

Joel Mokyr. Chapter 17 - Long-Term Economic Growth and the History of Technology. In *Handbook of Economic Growth*, volume 1, Part B, pages 1113–1180. Elsevier, 2005. (Cited on page 138.)

Laura Moreno, Wathsala Bandara, Sonia Haiduc, and Andrian Marcus. On the Relationship between the Vocabulary of Bug Reports and Source Code. In *Proceedings 29th International Conference on Software Maintenance (ICSM)*, pages 452–455, Eindhoven, The Netherlands, 2013. IEEE. (Cited on page 216.)

Henry Muccini, Antonia Bertolino, and Paola Inverardi. Using software architecture for code testing. *Transactions on Software Engineering*, 30(3): 160–171, 2004. (Cited on pages 3 and 91.)

Henry Muccini, Marcio Dias, and Debra J. Richardson. Software architecture-based regression testing. *Journal of Systems and Software*, 79(10):1379–1396, 2006. (Cited on page [234](#).)

Rick Mugridge. Managing Agile Project Requirements with Storytest-Driven Development. *IEEE Software*, 25(1):68–75, 2008. (Cited on pages [111](#) and [185](#).)

Gail C. Murphy, David Notkin, and Kevin J. Sullivan. Software reflexion models: bridging the gap between design and implementation. *Transactions on Software Engineering*, 27(4):364–380, 2001. (Cited on pages [3](#) and [91](#).)

John Mylopoulos and Jaelson Castro. Tropos: A Framework for Requirements-Driven Software Development. In *Information systems engineering: state of the art and research themes*, Lecture Notes in Computer Science. Springer Verlag, 2000. (Cited on page [108](#).)

Mika V. Mäntylä and Juha Itkonen. How are software defects found? The role of implicit defect detection, individual responsibility, documents, and knowledge. *Information and Software Technology*, 56(12):1597–1612, 2014. (Cited on page [22](#).)

Justus D. Naumann and A. Milton Jenkins. Prototyping: The New Paradigm for Systems Development. *MIS Quarterly*, 6(3):29–44, 1982. (Cited on pages [2](#) and [91](#).)

Ismael Navarro, Nancy Leveson, and Kristina Lunqvist. Semantic decoupling: Reducing the impact of requirement changes. *Requirements Engineering*, 15(4):419–437, 2010. (Cited on page [120](#).)

Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-Marc Jézéquel. A requirement-based approach to test product families. In *Proceedings 5th International Workshop on Software Product-Family Engineering (PFE)*, pages 198–210, Siena, Italy, 2004. Springer. (Cited on page [112](#).)

Clémentine Nebut, Franck Fleurey, Yves Le Traon, and Jean-Marc Jézéquel. Automatic Test Generation: A Use Case Driven Approach. *Transactions on Software Engineering*, 32(3):140–155, 2006. (Cited on pages [42](#), [86](#), [109](#), and [117](#).)

Sridhar Nerur, RadhaKanta Mahapatra, and George Mangalaraj. Challenges of Migrating to Agile Methodologies. *Communications of the ACM*, 48(5):72–78, 2005. (Cited on page [171](#).)

David Newman, Jey H. Lau, Karl Grieser, and Timothy Baldwin. Automatic Evaluation of Topic Coherence. In *Human Language Technologies*:

*The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 100–108, Los Angeles, USA, 2010. Association for Computational Linguistics. (Cited on pages [239](#), [244](#), and [253](#).)

Anh T. Nguyen, Tung T. Nguyen, Jafar Al-Kofahi, Hung V. Nguyen, and Tien N. Nguyen. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Proceedings 26th International Conference on Automated Software Engineering (ASE)*, pages 263–272, Lawrence, USA, 2011. IEEE. (Cited on page [240](#).)

Mahmood Niazi, David Wilson, and Didar Zowghi. Critical success factors for software process improvement implementation: an empirical study. *Software Process: Improvement and Practice*, 11(2):193–211, 2006. (Cited on page [145](#).)

Nan Niu, Yijun Yu, Bruno González-Baixaulli, Neil Ernst, Julio Cesar Sam-pao Do Prado Leite, and John Mylopoulos. Aspects across Software Life Cycle: A Goal-Driven Approach. In *Transactions on Aspect-Oriented Software Development*, volume 5560 of *Lecture Notes in Computer Science*, pages 83–110. Springer, 2009. (Cited on page [111](#).)

Zhenxing Niu, Gang Hua, Xinbo Gao, and Qi Tian. Context aware topic model for scene recognition. In *Proceedings Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2743–2750, Providence, USA, 2012. IEEE. (Cited on page [237](#).)

Dan North. Behavior Modification. *Better Software*, (3), 2006. (Cited on page [177](#).)

Nur Nurmuliani, Didar Zowghi, and Sue Fowell. Analysis of requirements volatility during software development life cycle. In *Proceedings 4th Australian Software Engineering Conference (ASWEC)*, pages 28–37, Melbourne, Australia, 2004. IEEE. (Cited on page [1](#).)

Bashar Nuseibeh and Steve Easterbrook. Requirements Engineering: A Roadmap. In *Proceedings 22nd International Conference on Software Engineering, Future of Software Engineering Track (ICSE)*, pages 35–46, Limerick, Ireland, 2000. ACM. (Cited on page [137](#).)

Rocco Oliveto, Malcolm Gethers, Denys Poshyvanyk, and Andrea De Lucia. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *Proceedings 18th International Conference on Program Comprehension (ICPC)*, pages 68–71, Braga, Portugal, 2010. IEEE. (Cited on page [211](#).)

George Ostrouchov, Wei-Chen Chen, Drew Schmidt, and Pragneshkumar Patel. Programming with Big Data in R, 2012. URL <http://r-pbd.org/>. (Cited on page 218.)

Oxford English Dictionary. medium, n., December 2011. URL <http://www.oed.com/viewdictionaryentry/Entry/115772>. (Cited on page 99.)

Federica Paci, Fabio Massacci, Fabrice Bouquet, and Stephane Debricon. Managing Evolution by Orchestrating Requirements and Testing Engineering Processes. In *Proceedings 5th International Conference on Software Testing, Verification and Validation (ICST)*, pages 834–841, Montreal, Canada, 2012. IEEE. (Cited on pages 38 and 83.)

Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. How to Effectively Use Topic Models for Software Engineering Tasks? An Approach Based on Genetic Algorithms. In *Proceedings 35th International Conference on Software Engineering (ICSE)*, pages 522–531, San Francisco, USA, 2013. IEEE. (Cited on pages 197, 210, and 239.)

Michael C. Panis. Successful Deployment of Requirements Traceability in a Commercial Engineering Organization...Really. In *Proceedings 18th Requirements Engineering Conference (RE)*, pages 303–307, Sydney, Australia, 2010. IEEE. (Cited on page 32.)

Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent Semantic Indexing: A Probabilistic Analysis. In *Proceedings 7th Symposium on Principles of Database Systems*, pages 159–168, Seattle, USA, 1998. ACM. (Cited on page 237.)

Raja Parasuraman, Thomas B. Sheridan, and Christopher D. Wickens. A model for types and levels of human interaction with automation. *Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 30(3):286–297, 2000. (Cited on pages 234, 240, 241, and 242.)

Shelly Park and Frank Maurer. Communicating Domain Knowledge in Executable Acceptance Test Driven Development. In *Proceedings 10th Internal Conference on Agile Processes in Software Engineering and Extreme Programming (XP)*, pages 23–32, Pula, Italy, 2009. Springer. (Cited on pages 183 and 186.)

Shelly Park and Frank Maurer. A Literature Review on Story Test Driven Development. In *Proceedings 11th International Conference on Agile Software Development (XP)*, pages 208–213, Trondheim, Norway, 2010. Springer. (Cited on pages 177, 185, and 186.)

Judea Pearl. Why There Is No Statistical Test For Confounding, Why Many Think There Is, And Why They Are Almost Right. *Department of Statistics, UCLA*, 1998. (Cited on page [13](#).)

Gilles Perrouin, Sagar Sen, Jacques Klein, Benoit Baudry, and Yves Le Traon. Automated and Scalable T-wise Test Case Generation Strategies for Software Product Lines. In *Proceedings 3rd International Conference on Software Testing, Verification and Validation (ICST)*, pages 459–468, Paris, France, 2010. IEEE. (Cited on page [193](#).)

Kai Petersen and Claes Wohlin. Context in Industrial Software Engineering Research. In *Proceedings 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 401–404, Orlando, USA, 2009. IEEE. (Cited on page [104](#).)

Kai Petersen and Claes Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15(6):654–693, 2010. (Cited on pages [5](#) and [142](#).)

Kai Petersen, Cigdem Gencel, Negin Asghari, Dejan Baca, and Stefanie Betz. Action Research As a Model for Industry-academia Collaboration in the Software Engineering Context. In *Proceedings International Workshop on Long-term Industrial Collaboration on Software Engineering (WISE)*, pages 55–62, Västerås, Sweden, 2014. ACM. (Cited on page [18](#).)

Nathan H. Petschenik. Practical Priorities in System Testing. *IEEE Software*, 2(5):18–23, 1985. (Cited on page [233](#).)

Fredrik Pettersson, Martin Ivarsson, Tony Gorschek, and Peter Öhman. A practitioner's guide to light weight software process assessment and improvement planning. *Journal of Systems and Software*, 81(6):972–995, 2008. (Cited on pages [49](#), [145](#), [191](#), and [247](#).)

Shari Lawrence Pfleeger and Joanne M. Atlee. *Software Engineering: Theory and Practice*. Prentice Hall, 4 edition, 2009. (Cited on pages [2](#) and [91](#).)

Simon Pickin, Claude Jard, Tierry Jéron, Jean-Marc Jézéquel, and Yves Le Traon. Test Synthesis from UML Models of Distributed Software. *Transactions on Software Engineering*, 33(4):252 –269, 2007. (Cited on pages [114](#) and [121](#).)

Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008. (Cited on page [34](#).)

Paulo F. Pires, Flávia C. Delicato, Raphael Cóbé, Thais Batista, Jospeh G. Davis, and Joo Hee Song. Integrating ontologies, model driven, and CNL in a multi-viewed approach for requirements engineering. *Requirements Engineering*, 16(2):133–160, 2011. (Cited on page [120](#).)

Klaus Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, Berlin, Germany, 1st edition, 2010. (Cited on page [176](#).)

Klaus Pohl and Andreas Metzger. Software Product Line Testing. *Communications of the ACM*, 49(12):78–81, 2006. (Cited on page [193](#).)

Denys Poshyvanyk and Adrian Marcus. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. In *Proceedings 15th International Conference on Program Comprehension (ICPC)*, pages 37–48, Banff, Canada, 2007. IEEE. (Cited on pages [213](#) and [216](#).)

Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Adrian Marcus, Giuliano Antoniol, and Václav Rajlich. Combining Probabilistic Ranking and Latent Semantic Indexing for Feature Identification. In *Proceedings 14th International Conference on Program Comprehension (ICPC)*, pages 137–148, Athens, Greece, 2006. IEEE. (Cited on pages [196](#) and [211](#).)

Denys Poshyvanyk, Malcolm Gethers, and Adrian Marcus. Concept location using formal concept analysis and information retrieval. *Transactions on Software Engineering and Methodology*, 21(4), 2012. (Cited on pages [196](#), [216](#), and [229](#).)

Hendrik Post, Carsten Sinz, Florian Merz, Thomas Gorges, and Thomas Kropf. Linking Functional Requirements and Software Verification. In *Proceedings 17th International Requirements Engineering Conference (RE)*, pages 295–302, Atlanta, USA, 2009. IEEE. (Cited on pages [22](#), [30](#), [41](#), [90](#), and [92](#).)

Abdallah Qusef, Gabriele Bavota, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. Recovering test-to-code traceability using slicing and textual analysis. *Journal of Systems and Software*, 88:147–168, 2014. (Cited on pages [32](#), [34](#), [196](#), [197](#), [198](#), [228](#), and [232](#).)

Balasubramaniam Ramesh. Factors Influencing Requirements Traceability Practice. *Communications of the ACM*, 41(12):37–44, 1998. (Cited on page [41](#).)

Balasubramaniam Ramesh and Matthias Jarke. Toward reference models for requirements traceability. *Transactions on Software Engineering*, 27(1): 58–93, 2001. (Cited on pages [33](#), [92](#), and [93](#).)

- Balasubramaniam Ramesh, Curtis Stubbs, Timothy Powers, and Michael Edwards. Requirements traceability: Theory and practice. *Annals of Software Engineering*, 3(1):397–415, 1997. (Cited on pages [6](#) and [41](#).)
- Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010. (Cited on pages [175](#), [177](#), [183](#), [184](#), and [185](#).)
- Brian Randell. Towards a methodology of computing system design. In *NATO Software Engineering Conference*, pages 204–208, Garmisch, Germany, 1968. (Cited on pages [41](#) and [93](#).)
- Felix Redmill. Theory and practice of risk-based testing. *Software Testing, Verification and Reliability*, 15(1):3–20, 2005. (Cited on page [244](#).)
- Björn Regnell and Per Runeson. Combining Scenario-based Requirements with Static Verification and Dynamic Testing. In *Proceedings 4th Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 195–206, Pisa, Italy, 1998. (Cited on page [42](#).)
- Björn Regnell, Per Runeson, and Claes Wohlin. Towards integration of use case modelling and usage-based testing. *Journal of Systems and Software*, 50(2):117–130, 2000. (Cited on pages [22](#) and [42](#).)
- Björn Regnell, Richard Berntsson Svensson, and Krzysztof Wnuk. Can We Beat the Complexity of Very Large-Scale Requirements Engineering? In *Proceedings 14th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 123–128, Montpellier, France, 2008. Springer. (Cited on page [1](#).)
- Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato, and Paolo Tonella. Using acceptance tests as a support for clarifying requirements: A series of experiments. *Information and Software Technology*, 51(2):270–283, 2009. (Cited on page [84](#).)
- Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-researchers*. John Wiley & Sons, 2nd edition, 2002. (Cited on pages [10](#), [11](#), [13](#), [14](#), [15](#), [18](#), [22](#), [42](#), [47](#), [53](#), and [112](#).)
- Gretchen B. Rossman and Bruce L. Wilson. Numbers and Words Combining Quantitative and Qualitative Methods in a Single Large-Scale Evaluation Study. *Evaluation Review*, 9(5):627–643, 1985. (Cited on page [11](#).)
- Gregg Rothermel and Mary J. Harrold. Analyzing regression test selection techniques. *Transactions on Software Engineering*, 22(8):529–551, 1996. (Cited on pages [3](#), [190](#), [193](#), and [233](#).)

Terence P. Rout, Khaled El Emam, Mario Fusani, Dennis Goldenson, and Ho-Won Jung. SPICE in retrospect: Developing a standard for process assessment. *Journal of Systems and Software*, 80(9):1483–1493, 2007. (Cited on page 146.)

Per Runeson. A survey of unit testing practices. *IEEE Software*, 23(4):22–29, 2006. (Cited on page 176.)

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. (Cited on pages 11, 13, 15, 21, 51, and 253.)

Per Runeson, Carina Andersson, and Martin Höst. Test processes in software product evolution—a qualitative survey on the state of practice. *Journal of Software Maintenance and Evolution: Research and Practice*, 15(1):41–59, 2003. (Cited on page 86.)

Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley & Sons, Hoboken, USA, 1st edition, 2012. (Cited on pages 12, 42, 43, 45, 47, 49, 53, 179, 182, and 183.)

Ioana Rus and Mikael Lindvall. Knowledge Management in Software Engineering. *IEEE Software*, 19(3):26–38, 2002. (Cited on page 139.)

Giedre Sabaliauskaite, Annabella Loconsole, Emelie Engström, Michael Unterkalmsteiner, Björn Regnell, Per Runeson, Tony Gorschek, and Robert Feldt. Challenges in aligning requirements engineering and verification in a large-scale industrial context. In *Proceedings 16th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 128–142, Essen, Germany, 2010. Springer. (Cited on pages 38, 39, 43, 87, 88, 90, and 92.)

Gerard Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975. (Cited on pages 28, 30, 201, and 237.)

Philip Samuel, Rajib Mall, and Pratyush Kanth. Automatic test case generation from UML communication diagrams. *Information and Software Technology*, 49(2):158–171, 2007. (Cited on pages 3, 91, and 115.)

Walt Scacchi. Process Models in Software Engineering. In *Encyclopedia of Software Engineering*, pages 993–1005. John Wiley & Sons, 2001. (Cited on page 89.)

Kurt Schneider, Kai Stapel, and Eric Knauss. Beyond Documents: Visualizing Informal Communication. In *Proceedings 3rd International Workshop*

*on Requirements Engineering Visualization*, pages 31–40, Barcelona, Spain, 2008. IEEE. (Cited on pages 5 and 141.)

Ken Schwaber. SCRUM Development Process. In *Business Object Design and Implementation*, pages 117–134. Springer London, 1997. (Cited on page 146.)

Robert Seater, Daniel Jackson, and Rohit Gheyi. Requirement progression in problem frames: Deriving specifications from requirements. *Requirements Engineering*, 12(2):77–102, 2007. (Cited on page 114.)

SEI. Appraisal Requirements for CMMI, Version 1.2 (ARC, V1.2). Technical Report CMU/SEI-2006-TR-011, Software Engineering Institute, Carnegie Mellon, Pittsburgh, USA, 2006. URL <http://www.sei.cmu.edu/library/abstracts/reports/06tr011.cfm>. (Cited on page 146.)

Helen Sharp, Hugh Robinson, and Marian Petre. The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers*, 21(1–2):108–116, 2009. (Cited on page 33.)

Martin Shepperd, David Bowes, and Tracy Hall. Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *Transactions on Software Engineering*, 40(6):603–616, 2014. (Cited on page 190.)

Mark Sherriff, Mike Lake, and Laurie Williams. Prioritization of Regression Tests using Singular Value Decomposition with Empirical Change Records. In *Proceedings 18th International Symposium on Software Reliability (ISSRE)*, pages 81–90, Trollhättan, Sweden, 2007. IEEE. (Cited on page 236.)

David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures, Second Edition*. Chapman and Hall/CRC, Boca Raton, 2nd edition edition, 2000. (Cited on pages 228, 229, and 230.)

Forrest Shull, Janice Singer, and Dag I. K. Sjøberg, editors. *Guide to Advanced Empirical Software Engineering*. Springer, London, 2008. (Cited on page 11.)

Sebastian Siegl, Kai-Steffen Hielscher, and Reinhard German. Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models. In *Proceedings 18th International Conference on Requirements Engineering (RE)*, pages 345–350, Sydney, Australia, 2010. IEEE. (Cited on pages 92, 109, and 117.)

Ernst Sikora, Bastian Tenbergen, and Klaus Pohl. Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering*, 17(1):57–78, 2012. (Cited on page 84.)

Igor Siveroni, Andrea Zisman, and George Spanoudakis. A UML-based static verification framework for security. *Requirements Engineering*, 15(1):95–118, 2010. (Cited on page [114](#).)

Dag I. K. Sjøberg, Tore Dybå, and Magne Jørgensen. The Future of Empirical Methods in Software Engineering Research. In *Proceedings Workshop on the Future of Software Engineering (FOSE)*, pages 358–378, Minneapolis, USA, 2007. IEEE. (Cited on page [94](#).)

Dag I. K. Sjøberg, Tore Dybå, Bente C. D. Anda, and Jo E. Hannay. Building Theories in Software Engineering. In *Guide to Advanced Empirical Software Engineering*, pages 312–336. Springer London, 2008. (Cited on pages [35](#) and [36](#).)

Mark D. Smucker, James Allan, and Ben Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings 16th Conference on Information and Knowledge Management (CIKM)*, pages 623–632, Lisbon, Portugal, 2007. ACM. (Cited on pages [228](#) and [230](#).)

Ian Sommerville. Integrated requirements engineering: a tutorial. *IEEE Software*, 22(1):16–23, 2005. (Cited on page [175](#).)

George Spanoudakis and Andrea Zisman. Software Traceability: A Roadmap. In *Handbook of Software Engineering and Knowledge Engineering*, pages 395–428. World Scientific Publishing, 2004. (Cited on page [6](#).)

Kai Stapel, Kurt Schneider, Daniel Lübke, and Thomas Flohr. Improving an Industrial Reference Process by Information Flow Analysis: A Case Study. In *Proceedings 8th International Conference on Product-Focused Software Process Improvement (PROFES)*, pages 147–159, Riga, Latvia, 2007. Springer. (Cited on pages [5](#) and [141](#).)

Kai Stapel, Eric Knauss, Kurt Schneider, and Nico Zazwora. FLOW Mapping: Planning and Managing Communication in Distributed Teams. In *Proceedings 6th International Conference on Global Software Engineering (ICGSE)*, pages 190–199, Helsinki, Finland, 2011. IEEE. (Cited on pages [5](#), [26](#), and [141](#).)

George E Stark, Paul Oman, Alan Skillicorn, and Alan Ameele. An examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, 11(5):293–309, 1999. (Cited on page [1](#).)

Christoph J. Stettina and Werner Heijstek. Necessary and Neglected?: An Empirical Study of Internal Documentation in Agile Software Development Teams. In *Proceedings 29th International Conference on Design of Communication (SIGDOC)*, pages 159–166, Pisa, Italy, 2011. ACM. (Cited on page [171](#).)

- Mark Steyvers and Thomas Griffiths. Probabilistic topic models. In *Handbook of Latent Semantic Analysis*. Psychology Press, 1st edition, 2007. (Cited on pages 196, 234, 237, 239, 243, 250, and 254.)
- Diane E. Strode, Sid L. Huff, Beverley Hope, and Sebastian Link. Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6):1222–1238, 2012. (Cited on page 33.)
- Mikael Svahnberg, Jilles van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Software: Practice and Experience*, 35(8):705–54, 2005. (Cited on page 94.)
- Mikael Svahnberg, Tony Gorschek, Thi Than Loan Nguyen, and Mai Nguyen. Uni-REPM: a framework for requirements engineering process assessment. *Requirements Engineering*, 20(1):91–118, 2015. (Cited on page 145.)
- Jian Tang, Zhaoshi Meng, Xuanlong Nguyen, Qiaozhu Mei, and Ming Zhang. Understanding the Limiting Factors of Topic Modeling via Posterior Contraction Analysis. In *Proceedings 31st International Conference on Machine Learning (ICML)*, pages 190–198, Beijing, China, 2014. (Cited on pages 245 and 254.)
- Maria Taramigkou, Eftimios Bothos, Konstantinos Christidis, Dimitris Apostolou, and Gregoris Mentzas. Escape the Bubble: Guided Exploration of Music Preferences for Serendipity and Novelty. In *Proceedings 7th Conference on Recommender Systems*, pages 335–338, Hong Kong, China, 2013. ACM. (Cited on page 234.)
- Gregory Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report 7007.011, National Institute of Standards and Technology, Gaithersburg, USA, 2002. (Cited on page 1.)
- Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006. (Cited on page 231.)
- The Apache Software Foundation. Apache Lucene - Apache Lucene Core, 2014. URL <http://lucene.apache.org/core/>. (Cited on page 210.)
- Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the Evolution of Topics in Source Code Histories. In *Proceedings 8th Working Conference on Mining Software Repositories (MSR)*, pages 173–182, Honolulu, USA, 2011. ACM. (Cited on page 240.)
- Stephen W. Thomas, Meiyappan Nagappan, Dorothea Blostein, and Ahmed E. Hassan. The Impact of Classifier Configuration and Classifier

Combination on Bug Localization. *Transactions on Software Engineering*, 39(10):1427–1443, 2013. (Cited on pages 210 and 229.)

Stephen W. Thomas, Hadi Hemmati, Ahmed E. Hassan, and Dorothea Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212, 2014. (Cited on pages 197 and 240.)

Christer Thörn. Current state and potential of variability management practices in software-intensive SMEs: Results from a regional industrial survey. *Information and Software Technology*, 52(4):411–421, 2010. (Cited on pages 191, 193, and 247.)

Michael Unterkalmsteiner, Tony Gorschek, A. K. M. Moinul Islam, Chow Kian Cheng, Rahadian Bayu Permadi, and Robert Feldt. Evaluation and Measurement of Software Process Improvement - A Systematic Literature Review. *Transactions on Software Engineering*, 38(2):398–424, 2012. (Cited on page 117.)

Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek. A Taxonomy for Requirements Engineering and Software Test Alignment. *Transactions on Software Engineering and Methodology*, 23(2), 2014a. (Cited on pages 1, 139, 142, 145, 147, 151, 161, and 168.)

Michael Unterkalmsteiner, Tony Gorschek, and Robert Feldt. Supplementary Material to "Large-scale Information Retrieval - an experience report from industrial application", 2014b. URL <http://www.bth.se/com/mun.nsf/pages/autotcs-exp>. (Cited on pages 203 and 210.)

Michael Unterkalmsteiner, Tony Gorschek, Robert Feldt, and Niklas Laveson. Large-scale Information Retrieval in Software Engineering - An Experience Report from Industrial Application. *Empirical Software Engineering*, 2015. Under Review. (Cited on page 234.)

Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5):297–312, 2012. (Cited on pages 92, 94, and 193.)

Eero J. Uusitalo, Marko Komssi, Marjo Kauppinen, and Alan M. Davis. Linking Requirements and Testing in Practice. In *Proceedings 16th International Conference on Requirements Engineering (RE)*, pages 265–270, Catalunya, Spain, 2008. IEEE. (Cited on pages 22, 30, 33, 38, 40, 41, 83, 84, 87, 88, 90, 92, 93, and 185.)

Engin Uzuncaova, Sarfraz Khurshid, and Don Batory. Incremental Test Generation for Software Product Lines. *Transactions on Software Engineering*, 36(3):309–322, 2010. (Cited on page 121.)

- Pedro Valderas and Vicente Pelechano. Introducing requirements traceability support in model-driven development of web applications. *Information and Software Technology*, 51(4):749–768, 2009. (Cited on page [115](#).)
- Andrew H. van de Ven, André L. Delbecq, and Richard Koenig, Jr. Determinants of Coordination Modes within Organizations. *American Sociological Review*, 41(2):322–338, 1976. (Cited on pages [9](#), [24](#), and [26](#).)
- Axel van Lamsweerde. Formal Specification: A Roadmap. In *Proceedings 22nd International Conference on Software Engineering, Future of Software Engineering Track (ICSE)*, pages 147–159, Limerick, Ireland, 2000. ACM. (Cited on page [176](#).)
- Sira Vegas, Natalia Juristo, and Victor R. Basili. Maturing Software Engineering Knowledge through Classifications: A Case Study on Unit Testing Techniques. *Transactions on Software Engineering*, 35(4):551–565, 2009. (Cited on page [94](#).)
- June M. Verner and William M. Evanco. In-house software development: what project management practices lead to success? *IEEE Software*, 22(1): 86–93, 2005. (Cited on pages [26](#) and [138](#).)
- Hanna M. Wallach, David M. Mimno, and Andrew McCallum. Rethinking LDA: Why Priors Matter. In *Proceedings 23rd Annual Conference on Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2009a. Curran Associates, Inc. (Cited on pages [243](#) and [249](#).)
- Hanna M. Wallach, Iain Murray, Ruslan Salakhutdinov, and David M. Mimno. Evaluation Methods for Topic Models. In *Proceedings 26th Annual International Conference on Machine Learning (ICML)*, pages 1105–1112, Montreal, Canada, 2009b. ACM. (Cited on pages [239](#) and [243](#).)
- Shuai Wang, Arnaud Gotlieb, Shaukat Ali, and Marius Liaaen. Automated Test Case Selection Using Feature Model: An Industrial Case Study. In *Proceedings 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, pages 237–253, Miami, USA, 2013. Springer. (Cited on pages [234](#), [237](#), and [241](#).)
- Robert Watkins and Mark Neal. Why and how of requirements tracing. *IEEE Software*, 11(4):104–106, 1994. (Cited on pages [38](#), [41](#), [86](#), and [93](#).)
- Xing Wei and W. Bruce Croft. LDA-based Document Models for Ad-hoc Retrieval. In *Proceedings 29th Annual International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 178–185, Seattle, USA, 2006. ACM. (Cited on page [237](#).)
- James A. Whittaker. What is software testing? And why is it so hard? *IEEE Software*, 17(1):70–79, 2000. (Cited on pages [176](#) and [233](#).)

Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, Berlin, Germany, 1st edition, 2014a. (Cited on pages [16](#), [17](#), [18](#), [25](#), and [142](#).)

Roel Wieringa. Empirical research methods for technology validation: Scaling up to practice. *Journal of Systems and Software*, 95:19–31, 2014b. (Cited on pages [19](#) and [142](#).)

Roel Wieringa and Ayşe Morali. Technical Action Research As a Validation Method in Information Systems Design Science. In *Proceedings 7th International Conference on Design Science Research in Information Systems and Technology (DESRIST)*, pages 220–238, Las Vegas, USA, 2012. Springer. (Cited on pages [18](#) and [144](#).)

Greger Wikstrand, Robert Feldt, Jeevan K. Gorantla, Wang Zhe, and Conor White. Dynamic Regression Test Selection Based on a File Cache An Industrial Evaluation. In *Proceedings 2nd International Conference on Software Testing Verification and Validation (ICST)*, pages 299–302, Denver, USA, 2009. IEEE. (Cited on page [237](#).)

Kristina Winbladh, Thomas A. Alspaugh, Hadar Ziv, and Debra J. Richardson. An Automated Approach for Goal-driven, Specification-based Testing. In *Proceedings 21st International Conference on Automated Software Engineering (ASE)*, pages 289–292, Tokyo, Japan, 2006. IEEE. (Cited on page [112](#).)

Claes Wohlin. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings 18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10, London, UK, 2014. ACM. (Cited on page [21](#).)

Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, 2000. (Cited on pages [10](#), [12](#), [13](#), [19](#), [21](#), [115](#), [202](#), and [203](#).)

Claes Wohlin, Aybuke Aurum, Lefteris Angelis, Laura Phillips, Yvonne Dittrich, Tony Gorschek, Hakan Grahn, Kennet Hennigsson, Simon Kagstrom, Graham Low, Per Rovegard, Piotr Tomaszewski, Christine van Toorn, and Jeff Winter. The Success Factors Powering Industry-Academia Collaboration. *IEEE Software*, 29(2):67–73, 2012. (Cited on pages [145](#) and [189](#).)

Xin Xia, David Lo, Xinyu Wang, and Bo Zhou. Dual analysis for recommending developers to resolve bugs. *Journal of Software: Evolution and Process*, 2015. In Print. (Cited on pages [34](#), [228](#), and [232](#).)

- Dianxiang Xu, Omar El-Ariss, Weifeng Xu, and Linzhang Wang. Testing aspect-oriented programs with finite state machines. *Software Testing, Verification and Reliability*, 22(4):267–293, 2012. (Cited on page 120.)
- Zhiwei Xu, Kehan Gao, and Taghi M. Khoshgoftaar. Application of fuzzy expert system in test case selection for system regression test. In *Proceedings International Conference on Information Reuse and Integration (IRI)*, pages 120–125, Las Vegas, USA, 2005. IEEE. (Cited on page 236.)
- Robert K. Yin. *Case Study Research: Design and Methods*. Sage Publications, third edition. edition, 2003. (Cited on pages 3, 10, and 13.)
- Shin Yoo and Mark Harman. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability*, 22(2):67–120, 2012. (Cited on pages 5, 193, 197, 233, 234, and 236.)
- Tao Yue, Lionel C. Briand, and Yvan Labiche. A systematic review of transformation approaches between user requirements and analysis models. *Requirements Engineering*, 16(2):75–99, 2011. (Cited on page 42.)
- yWorks. yEd - Graph Editor, 2014. URL [http://www.yworks.com/en/products\\_yed\\_about.html](http://www.yworks.com/en/products_yed_about.html). (Cited on page 153.)
- Dimitrios Zeimpekis and Efstratios Gallopoulos. TMG: A MATLAB Toolbox for Generating Term-Document Matrices from Text Collections. In *Grouping Multidimensional Data*, pages 187–210. Springer, Berlin, Heidelberg, 2006. (Cited on page 210.)
- Yuan Cao Zhang, Diarmuid Séaghdha, Daniele Quercia, and Tamas Jambor. Auralist: Introducing Serendipity into Music Recommendation. In *Proceedings 5t International Conference on Web Search and Data Mining (WSDM)*, pages 13–22, Seattle, USA, 2012. ACM. (Cited on page 234.)
- Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. SNIAFL: Towards a static noninteractive approach to feature location. *Transactions on Software Engineering and Methodology*, 15(2):195–226, 2006. (Cited on page 196.)
- Joe Zou and Christopher J. Pavlovski. Control Cases during the Software Development Life-Cycle. In *Congress on Services - Part 1*, pages 337–344. IEEE, 2008. (Cited on page 109.)
- Radim Řehůřek. Subspace Tracking for Latent Semantic Analysis. In *Proceedings 33rd European Conference on Advances in Information Retrieval (ECIR)*, pages 289–300, Dublin, Ireland, 2011. Springer. (Cited on page 218.)

Darja Šmite, Claes Wohlin, Robert Feldt, and Tony Gorschek. Reporting Empirical Research in Global Software Engineering: A Classification Scheme. In *Proceedings 3rd International Conference on Global Software Engineering (ICGSE)*, pages 173 –181, Bangalore, India, 2008. IEEE. (Cited on page 121.)