Software Engineering

from Research and Practice Perspectives

Scientific Editors

Lech Madeyski
Mirosław Ochodek

Conferences organized by
the Polish Information Processing Society:

IX edition of the Congress of Young IT Scientists
XVI edition of the Polish Conference on Software Engineering
XXI edition of Real Time Systems

and co-organized by the Wielkopolska Branch of PIPS

XVIII edition of Signal Processing

were supported
by the Ministry of Science and Higher Education
within the program related to the implementation of tasks of science
dissemination
(Decision No. 1187/P-DUN/2014 on 07/07/2014)

Polish Information Processing Society

Software Engineering

from Research and Practice Perspectives

Scientific Editors

Lech Madeyski
Mirosław Ochodek

Poznan-Warsaw 2014

WYDAWNICTWO
Nakom ● Poznań

Authors

Dirk Riehle – ***CHAPTER 1***
Miklós Biró – ***CHAPTER 2***
Jakub Jurkiewicz, Piotr Kosiuczenko, Lech Madeyski, Mirosław Ochodek, Cezary Orłowski, Łukasz Radliński – ***CHAPTER 3***
Marek Majchrzak, Łukasz Stilger, Marek Matczak – ***CHAPTER 4***
Tomasz Sitek, Artur Ziółkowski – ***CHAPTER 5***
Robert Waszkowski – ***CHAPTER 6***
Jerzy Niepostyn, Andrzej Tyrowicz – ***CHAPTER 7***
Łukasz Radliński – ***CHAPTER 8***
Bogumiła Hnattkowska, Łukasz Wrona – ***CHAPTER 9***
Andrzej Ratkowski, Krzysztof Gawryś, Eliza Świątek – ***CHAPTER 10***
Ilona Bluemke, Anna Stępień – ***CHAPTER 11***
Michał Ćmil. Bartosz Walter – ***CHAPTER 12***
Wojciech Frącz, Jacek Dajda – ***CHAPTER 13***

Reviewers

Bartosz Alchimowicz, Piotr Czapiewski, Włodzimierz Dąbrowski, Iwona Dubielewicz, Bogumiła Hnatkowska, Jarosław Hryszko, Aleksander Jarzębowicz, Krzysztof Juszczyszyn, Sylwia Kopczyńska, Leszek Maciaszak, Michał Maćkowiak, Marek Majchrzak, Roland Manglus, Piotr Miklosik, Karolina Muszyńska, Michał Negacz, Mirosław Ochodek, Cezary Orłowski, Willy Picard, Łukasz Radliński, Jakub Rojek, Martin Shepperd, Michał Śmiałek, Janusz Sobecki, Mirosław Staron, Andrzej Stasiak, Wojciech Thomas, Lech Tuzinkiewicz, Tomasz Wala, Anita Walkowiak, Bartosz Walter, Jacek Widuch, Łukasz Wrona

Scientific Editors

Lech Madeyski
Mirosław Ochodek

ISBN 978-83-63919-16-0

Edition I
NAKOM Publishers, Starołęcka St. 18A, room 303, 61-361 Poznan, Poland

Table of Contents

Preface

Software Engineering is a relatively young engineering discipline. The term was coined during the NATO conference held in 1968 in Garmish, Germany. In fact, the composition of two words ``software" and ``engineering" was rather provocative at that time, because the young software development industry was going through difficult times – also known as the ``software crisis". When computers became more popular and software development went from scientific laboratories to massive production, it became obvious that without systematic, well founded ``engineering" methods, development of complex software systems would remain extremely difficult and risky business.

The IEEE Computer Society defines software engineering as: ``(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)." Thus, one could use the term *software engineer* either to refer to a person who professionally delivers software systems or to one who investigates phenomena related to software development – which seems natural to academia. In fact, the currently promoted agile software development methods show that effective software development organizations require both – practical skills of their employees and the ability to critically look at existing problems to continuously grow. Such thinking creates a unique opportunity for closer collaboration between practitioners and researches. However, before that could happen, both – engineers and researchers need to better understand each other's views on software development.

In this book, we asked practitioners and researchers to share their thoughts on chosen problems related to software development. We would like to begin the book by presenting selected, valuable results of research in the area of Software Engineering. Firstly, we would like to present two internationally recognized researchers – Dirk Riehle from the Friedrich-Alexander University of Erlangen-Nürnberg and Miklós Biró representing Software Competence Center Hagenberg, who changed the way we perceive Open Source and Software Process Improvement. Secondly, in Chapter 1, we discuss recent, prominent research in the area of Software Engineering conducted by researchers from the Polish research institutions. The rest of the book is organized into four parts: project management; requirements engineering; software architecture and design; and software quality.

In the first part of the book, the contributing authors discuss problems related to project management. In Chapter 2, the practitioners from Capgemini Poland share their practical experience from conducting projects in distributed environment using agile methodologies. In Chapter 3, the researchers from Gdansk University of Technology discus the possibility of supporting decision-making in IT projects.

In the second part of the book, different aspects of requirements engineering are discussed. In Chapter 4, Miklós Biró from Software Competence Center Hagenberg gives a brief overview of standard approaches to functional safety with examples from the medical domain. The following Chapter 5 was contributed by the authors combining perspectives – research (Warsaw University of Technology) and practice (Agencja Europejska). They discuss problems related to preserving consistency between semi-formal diagrams such as context diagrams and use-case diagrams. Finally, in Chapter 6, Łukasz Radliński, from West Pomeranian University of Technology investigates the factors that seem to influence user satisfaction in meeting business objectives and requirements.

The third part of the book is dedicated to software architecture and design. In Chapter 7, the researchers from Wroclaw University of Technology evaluate and compare three popular Enterprise Service Buses (ESB), based on the ISO 9126 standard. In Chapter 8, a group of researchers from Warsaw University of Technology discusses potential applications of architectural patterns to some of the emerging problems in so-called Internet of Things. In Chapter 9, the researchers from Warsaw University of Technology give advices on how to efficiently implement Data, Context and Interaction architectural pattern (DCI) in Java and C++ programming languages.

The last part of the book discusses problems related to software quality. In Chapter 10, two researchers from Poznan University of Technology present their extensible software tool E2A that allows defining and collecting software metrics. Finally, in Chapter 11, two authors from AGH University of Science and Technology present a promising approach to perform source code reviews on mobile devices.

Finally, we would like to express our deep gratitude towards the authors of chapters for their contributions and to the reviewers for their valuable remarks.

Lech Madeyski
Mirosław Ochodek

Dirk Riehle

Prof. Dr. Dirk Riehle is the Professor for Open Source Software at the Friedrich-Alexander University of Erlangen-Nürnberg. Before joining academia, Riehle led the Open Source Research Group at SAP Labs, LLC, in Palo Alto, California (Silicon Valley). Before this, he was the co-founder of an on-demand business software startup in Berlin, Germany, which used agile methods and strategically employed open source software. Riehle is interested in open source software engineering and agile methods, complexity science and human collaboration, as well as software design. Prof. Riehle holds a Ph.D. in computer science from ETH Zürich and an M.B.A. from Stanford Business School.

In more detail: Riehle's dissertation at ETH Zurich on object-oriented frameworks and design patterns explored the use of collaboration-based design (then called role modeling) to reduce complexity in the engineering of object-oriented software systems. It emphasized the use of design patterns in framework design and construction. He also translated the seminal Design Patterns book into German, all while employed at UBS' Ubilab, a Zurich-based industrial research lab, during the late 1990s. From 1999 to 2002, Riehle lead the design and implementation of the first UML virtual machine while employed at Skyva, a Boston-based software startup. The UML VM interpreted UML models as programs and made it faster, better, and cheaper to develop business applications. UML was treated as a framework for domain-specific languages for different aspects of business modeling. Skyva was acquired by ABB. After receiving an M.B.A. from Stanford Business School, Riehle co-founded a software startup in Berlin, Germany that provided on-demand software (SaaS) to small businesses. In 2006, he moved back to the United States to work for SAP in the Silicon Valley, where he was the principal investigator of open source and Web 2.0 applications research. In 2009 he moved to Germany for his current position as a professor at the University of Erlangen.

Riehle has published in leading journals and conferences, including the CACM, Computer, IEEE Software, OOPSLA, ICSE, and OSS. His publication record comprises more than 50 peer-reviewed and well-cited academic papers. He is serving on the editorial boards of TPLoP, IJOSSP and IJODE and he has been a reviewer for many leading journals, transactions, and conferences on object orientation and software engineering, including ACM TOSEM, IEEE TSE, OOPSLA, ECOOP, and OSS. He is the founder and chairman of the steering committee of the International Symposium on Wikis and Open Collaboration conference series and a founding member of the steering committee of the Onward! conference series. He is a member emeritus of the board and prior treasurer of the Hillside Group, the U.S.-based non-profit behind the software patterns community. He is a frequent speaker at academic conferences and colloquia and industry events alike.

Prof. Dr. Dirk Riehle is also frequently invited as a keynote speaker at recognized conferences, workshops, and seminars. Below, we present two selected abstracts of his research talks regarding Open Source.

Sustainable Open Source

- KKIO'14 Software Engineering Conference
- 2014 BITKOM Forum "Future of Open Source"
- Research Seminar at Victoria University of Wellington, New Zealand, 2012

Abstract: MySQL was sold for one billion US-dollar. Red Hat is worth a multiple of that. The Eclipse Foundation has pushed many software tool vendors out of business. How come that open source, a phenomenon dubbed "temporary" not only has become sustainable but the business strategy of choice? In this talk, I discuss the four main business models, two for-profit and two not-for-profit, that have made open source sustainable. These models are changing the business of software and the future of our industry.

Best of Our Empirical Open Source Research

- 39th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2013
- Research Seminar at Victoria University of Wellington, New Zealand, 2012

Abstract: Open source software is publicly developed software. Thus, for the first time, we can broadly analyze in data-driven detail how people program, how bugs come about, and how we could improve our tools. In this talk, I'll review six years of our open source empirical (data) research and highlight the most interesting insights, including how different (or not) open source is from closed source programming.

Miklós Biró

Dr. Miklós Biró is Key Researcher and Scientific Head of the Process and Quality Engineering Research Focus at Software Competence Center Hagenberg GmbH (SCCH, Austria), Full Professor (Ordentlicher Hochschulprofessor in German) nominated by the prime minister of Hungary, Doctor Habilitatus (Corvinus University of Budapest) with software engineering, university teaching (including professorship in the USA), research, and management experience.

Ph.D. in Mathematics (Loránd Eötvös University in Budapest), Master of Science in Management (Purdue University, USA). Fluent in Hungarian, English, and French. Initiating and managing role in numerous European projects. Author of Hungarian and English language books and publications.

Founding president of the professional division for Software Quality Management of the John von Neumann Computer Society, Hungarian national representative in IFIP TC-2 Software: Theory and Practice. Chair of professional conferences, member of programme committees and journal editorial board.

Dr. Miklós Biró contributed to the book as an author of Chapter 4, entitled "*Functional safety, traceability, and Open Services*."

PART I
PROJECT MANAGEMENT

Chapter 1

Recent Polish achievements in Software Engineering

Publications in top research journals (indexed by ISI) as well as citations are crucial in any research field to position the work and to build on the work of others. The objective of this chapter is twofold: to give an overview of the achievements of Polish research centers in the field of software engineering since 2010, and to present few recent contributions by researchers with Polish affiliations in ISI journals in the field of software engineering or closely related fields.

1.1. Introduction

Glass was the first who two decades ago published an assessment of systems and software engineering scholars and institutions [Gla94]. The set of journals selected by Glass included *IEEE Transactions on Software Engineering* (*TSE*), *ACM Transactions on Software Engineering and Methodologies* (*TOSEM*), *IEEE Software* (*SW*), *Information and Software Technology* (*IST*), *Journal of Systems and Software* (*JSS*), and *Software: Practice and Experience* (*SPE*). In 2009 Wong et al. [WTGBC09] have analyzed publications in the period of 2002–2006 using this set of journals extended by the *Empirical Software Engineering* (*EMSE*) journal to emphasize the importance of strong empirical component. The most recent report by Wong et al. was published in 2011 [WTGBC11].

A complementary series of analyses of the most cited articles in the software engineering journals has been published by Wohlin. The most recent of analyses was published in 2009 on a basis of 18 software engineering journals [Woh09].

However, to the best of our knowledge, neither similar analyses but related to Polish research institutions or researchers involved in the software engineering field, nor an overview of selected of contributions by researchers with Polish affiliations in ISI software engineering (or closely related) journals have been published so far. Hence, the aim of this chapter is to fill this gap.

The aim of the first section is to give an overview of the contribution of researchers with Polish affiliations in comparison to other European countries. The aim of the subsequent sections is to go into details and present short overviews of selected contributions of Polish authors published by ISI indexed journals within the software engineering field or computer science in general. The presented contributions include identification of events in use cases, solving the invariability problem in OCL, predicting the flow of defect correction effort to optimize the amount of quality assurance (QA) activities to minimize the total project effort, and model of a maturity capsule in software project management.

1.1.1. Selection decisions

An important decision when looking at Polish contributions to the field of software engineering is which ISI software engineering journals to include. We decided to include in the analyzed set of journals all of the software engineering journals analyzed by Wong et al. [WTGBC11] as well as Wohlin [Woh09], even if they changed their names (e.g., *Journal of Software: Evolution and Process* continues, since 2012, the tradition of the *Journal of Software Maintenance and Evolution: Research and Practice* and *Software Process: Improvements and Practice*, while *IET Software* continues, since 2007, the tradition of *IEE Proceedings - Software*). Then from the created superset of journals we excluded journals which are discontinued (*Annals of Software Engineering*, *Software Architecture*, *Software – Concepts and Tools*) or journals without impact factor in 2013. In spite of the fact that the created our set of journals based on the inclusion decisions of the renowned authors of previous analyses – Wong et al. [WTGBC11] and Wohlin [Woh09] – and minor constraints related to impact factor in year 2013, the created set of journals is by no means complete and can be extended even further. Our arbitrary decision is to extend the set of journals by adding the *Software and Systems Modeling* journal, which is strictly software engineering journal with assigned impact factor. There is also a wide range of computer science journals (e.g., *Computing and Informatics*, *Cybernetics and Systems: An International Journal*) which could be included on a paper by paper basis as some of them may be related to software engineering. However, it would need an extreme effort to check every published paper. As a result, the set of the analyzed journals is presented in Table 1.1.

All of these journals are indexed by Scopus, which provides an excellent search interface including ability to construct advanced search strings. The search string we used to constrain our search to the papers by authors with Polish affiliation published since 2010 in the aforementioned set of journals is presented below:

```
 ( ISSN(1049331X) OR ISSN(09288910) OR ISSN(13823256) OR ISSN(17518806) OR
ISSN(07407459) OR ISSN(00985589) OR ISSN(09505849) OR ISSN(02181940) OR
ISSN(1532060X) OR ISSN(09473602) OR ISSN(20477481) OR ISSN(09639314) OR
ISSN(09600833) OR ISSN(00380644) OR ISSN(01641212) OR ISSN(16191366)) AND
AFFIL(poland) AND (PUBYEAR > 2009)
```

Table 1.1. Set of analyzed software engineering journals.

ACM Transactions on Software Engineering and Methodology (TOSEM)
Automated Software Engineering (ASE)
Empirical Software Engineering (EMSE)
IET Software (IETSW)
IEEE Software (SW)
IEEE Transactions on Software Engineering (TSE)
Information and Software Technology (IST)
International Journal of Software Engineering and Knowledge Engineering (IJSEKE)
Journal of Software: Evolution and Process (JSEP)
Journal of Software Maintenance and Evolution: Research and Practice (JSME)
Requirements Engineering Journal (REJ)
Software and Systems Modeling (SoSyM)
Software Quality Journal (SQJ)
Software Testing, Verification and Reliability (STVR)
Software: Practice and Experience (SPE)
Journal of Systems and Software (JSS)

1.1.2. Search results

The search performed on 14 September 2014 returned 28 document results (sorted by number of citations):

1. L. Madeyski. "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment". In: *Information and Software Technology* 52.2 (2010), pp. 169–184. DOI: *10.1016/j.infsof.2009.08.007*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.08.007* – 19 citations
2. M. Ochodek, J. Nawrocki, and K. Kwarciak. "Simplifying Effort Estimation Based on Use Case Points". In: *Information and Software Technology* 53.3 (Mar. 2011), pp. 200–213. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2010.10.005*. URL: *http://dx.doi.org/10.1016/j.infsof.2010.10.005* – 16 citations
3. L. Madeyski and N. Radyk. "Judy – A Mutation Testing Tool for Java". In: *IET Software Journal (formerly IEE Proceedings Software)* 4.1 (2010). Draft: *http://madeyski.e-informatyka.pl/download/Madeyski10b.pdf*, pp. 32–42. DOI: *10.1049/iet-sen.2008.0038*. URL: *http://dx.doi.org/10.1049/iet-sen.2008.0038* – 16 citations
4. A. Janik and K. Zielinski. "AAOP-based Dynamically Reconfigurable Monitoring System". In: *Information and Software Technology* 52.4 (Apr. 2010), pp. 380–396. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2009.10.006*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.10.006* – 15 citations
5. G. J. Nalepa and K. Kluza. "UML REPRESENTATION FOR RULE-BASED APPLICATION MODELS WITH XTT2-BASED BUSINESS RULES". in: *International Journal of Software Engineering and Knowledge Engineering* 22.04 (2012), pp. 485–524. DOI: *10.1142/S021819401250012X*. URL: *http://www.worldscientific.com/doi/abs/10.1142/S021819401250012X* – 10 citations
6. M. Miłkowski. "Developing an Open-source, Rule-based Proofreading Tool". In: *Software:*

Practice and Experience 40.7 (June 2010), pp. 543–566. ISSN: 0038-0644. DOI: *10.1002/spe.v40:7*. URL: *http://dx.doi.org/10.1002/spe.v40:7* – 10 citations

7. A. Janik and K. Zielinski. "Adaptability Mechanisms for Autonomic System Implementation with AAOP". in: *Software: Practice and Experience* 40.3 (Mar. 2010), pp. 209–223. ISSN: 0038-0644. DOI: *10.1002/spe.v40:3*. URL: *http://dx.doi.org/10.1002/spe.v40:3* – 4 citations
8. P. Bachara, K. Blachnicki, and K. Zielinski. "Framework for Application Management with Dynamic Aspects J-EARS Case Study". In: *Information and Software Technology* 52.1 (Jan. 2010), pp. 67–78. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2009.06.003*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.06.003* – 4 citations
9. J. Floch et al. "A Comprehensive Engineering Framework for Guaranteeing Component Compatibility". In: *Journal of Systems and Software* 83.10 (Oct. 2010), pp. 1759–1779. ISSN: 0164-1212. DOI: *10.1016/j.jss.2010.04.075*. URL: *http://dx.doi.org/10.1016/j.jss.2010.04.075* – 3 citations
10. S. Deorowicz. "Solving Longest Common Subsequence and Related Problems on Graphical Processing Units". In: *Software: Practice and Experience* 40.8 (2010), pp. 673–700. ISSN: 0038-0644. DOI: *10.1002/spe.v40:8*. URL: *http://dx.doi.org/10.1002/spe.v40:8* – 3 citations
11. A. Zalewski and S. Kijas. "Beyond ATAM: Early Architecture Evaluation Method for Large-scale Distributed Systems". In: *Journal of Systems and Software* 86.3 (Mar. 2013), pp. 683–697. ISSN: 0164-1212. DOI: *10.1016/j.jss.2012.10.923*. URL: *http://dx.doi.org/10.1016/j.jss.2012.10.923* – 2 citations
12. K. Łukasiewicz and J. Miler. "Improving agility and discipline of software development with the Scrum and CMMI". in: *Software, IET* 6.5 (2012), pp. 416–422. ISSN: 1751-8806. DOI: *10.1049/iet-sen.2011.0193* – 2 citations
13. M. Janicki, M. Katara, and T. Pääkkönen. "Obstacles and Opportunities in Deploying Model-based GUI Testing of Mobile Software: A Survey". In: *Software Testing, Verification and Reliability* 22.5 (Aug. 2012), pp. 313–341. ISSN: 0960-0833. DOI: *10.1002/stvr.460*. URL: *http://dx.doi.org/10.1002/stvr.460* – 2 citations
14. M. Ochodek, B. Alchimowicz, J. Jurkiewicz, and J. Nawrocki. "Improving the Reliability of Transaction Identification in Use Cases". In: *Information and Software Technology* 53.8 (2011), pp. 885–897. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2011.02.004*. URL: *http://dx.doi.org/10.1016/j.infsof.2011.02.004* – 2 citations
15. R. Hofman. "Behavioral Economics in Software Quality Engineering". In: *Empirical Software Engineering* 16.2 (Apr. 2011), pp. 278–293. ISSN: 1382-3256. DOI: *10.1007/s10664-010-9140-x*. URL: *http://dx.doi.org/10.1007/s10664-010-9140-x* – 2 citations
16. J. Jurkiewicz, J. Nawrocki, M. Ochodek, and T. Głowacki. "HAZOP based identification of events in use cases". English. In: *Empirical Software Engineering* (2013), pp. 1–28. ISSN: 1382-3256. DOI: *10.1007/s10664-013-9277-5*. URL: *http://dx.doi.org/10.1007/s10664-013-9277-5* – 1 citation
17. A. Riel, A. Draghici, G. Draghici, D. Grajewski, and R. Messnarz. "Process and product innovation needs integrated engineering collaboration skills". In: *Journal of Software: Evolution and Process* 24.5 (2012), pp. 551–560. ISSN: 2047-7481. DOI: *10.1002/smr.497*. URL: *http://dx.doi.org/10.1002/smr.497* – 1 citation
18. P. Janczarek and J. Sosnowski. "Investigating software testing and maintenance reports: Case study". In: *Information and Software Technology* 0 (2014), pp. –. ISSN: 0950-5849. DOI:

http://dx.doi.org/10.1016/j.infsof.2014.06.015. URL: *http://www.sciencedirect.com/science/article/pii/S0950584914001542* – 0 citations

19. L. Madeyski and M. Jureczko. “Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study”. In: *Software Quality Journal* (accepted) (2014). DOI: *10.1007/s11219-014-9241-7*. URL: *http://dx.doi.org/10.1007/s11219-014-9241-7* – 0 citations
20. L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. “Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation”. In: *IEEE Transactions on Software Engineering* 40.1 (2014), pp. 23–42. ISSN: 0098-5589. DOI: *10.1109/TSE.2013.44*. URL: *http://dx.doi.org/10.1109/TSE.2013.44* – 0 citations
21. J. Sobecki. “Comparison of Selected Swarm Intelligence Algorithms in Student Courses Recommendation Application”. In: *International Journal of Software Engineering and Knowledge Engineering* 24.01 (2014), pp. 91–109 – 0 citations
22. B. Czarnacka-Chrobot. “RATIONALIZATION OF BUSINESS
SOFTWARE SYSTEMS DEVELOPMENT AND ENHANCEMENT PROJECTS INVESTMENT DECISIONS ON THE BASIS OF FUNCTIONAL SIZE MEASUREMENT”. in: *International Journal of Software Engineering and Knowledge Engineering* 23.06 (2013), pp. 839–868. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013500228* – 0 citations
23. T. Schulz, Ł. Radliński, T. Gorges, and W. Rosenstiel. “Predicting the Flow of Defect Correction Effort using a Bayesian Network Model”. English. In: *Empirical Software Engineering* 18.3 (2013), pp. 435–477. ISSN: 1382-3256. DOI: *10.1007/s10664-011-9175-7*. URL: *http://dx.doi.org/10.1007/s10664-011-9175-7* – 0 citations
24. P. Kosiuczenko. “Specification of Invariability in OCL”. in: *Software and Systems Modeling* 12.2 (May 2013), pp. 415–434. ISSN: 1619-1366. DOI: *10.1007/s10270-011-0215-y*. URL: *http://dx.doi.org/10.1007/s10270-011-0215-y* – 0 citations
25. W. Pedrycz. “KNOWLEDGE MANAGEMENT AND SEMANTIC MODELING: A ROLE OF INFORMATION GRANULARITY”. in: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 5–11. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013400019* – 0 citations
26. J. J. Jung, R. P. Katarzyniak, and N. T. Nguyen. “GUEST EDITORS; INTRODUCTION”. in: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 1–3. DOI: *10.1142/S0218194013020014* – 0 citations
27. R. P. Katarzyniak and G. Popek. “INTEGRATION OF MODAL AND FUZZY METHODS OF KNOWLEDGE REPRESENTATION IN ARTIFICIAL AGENTS”. in: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 13–29. DOI: *10.1142/S0218194013400020*. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013400020* – 0 citations
28. M. Psiuk, D. Żmuda, and K. Zielinski. “Distributed OSGi Built over Message-oriented Middleware”. In: *Software: Practice and Experience* 43.1 (Jan. 2013), pp. 1–31. ISSN: 0038-0644. DOI: *10.1002/spe.1148*. URL: *http://dx.doi.org/10.1002/spe.1148* – 0 citations

We repeated the search process for different countries (changing the part of the search string responsible for affiliation, e.g. from `AFFIL(poland)` into `AFFIL(germany)`) as well as the world (removing the part of the search string responsible for affiliation). Figure 1.1 presents a map of Europe including bubbles with bubble size proportional to the contribution of each country.

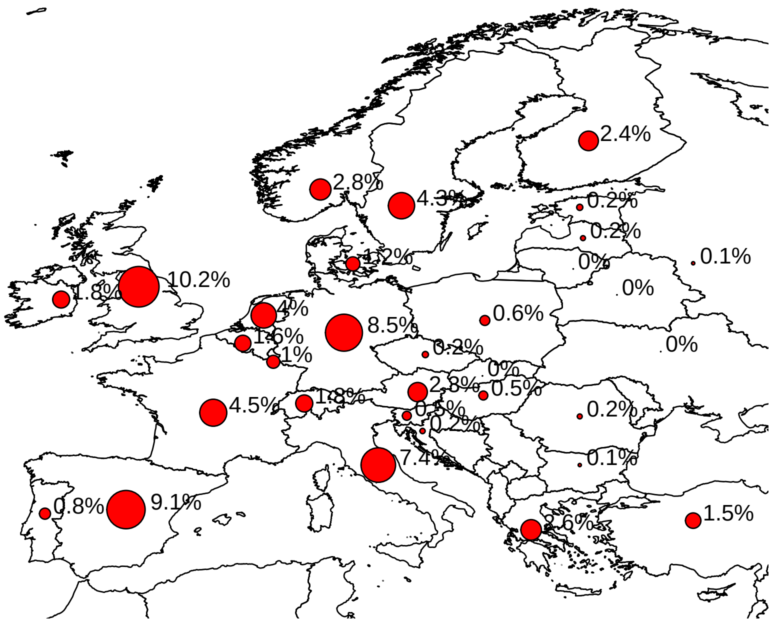


Figure 1.1. How European countries contribute to leading software engineering journals.

United Kingdom, Spain and Germany contribute the most in terms of the number of papers published in the analyzed set of leading software engineering journals and time frame, i.e. 10.2%, 9.1% and 8.5% respectively. It is worth mentioning that contribution of United States is about 21%.

Polish contribution (0.6%) is slightly less than the contribution of Portugal (0.8%), which cannot be considered an achievement taking into account that Poland is bigger than Portugal. Polish contribution is spread among 13 research institutions (e.g., Wroclaw University of Technology, AGH University of Technology, Poznan University of Technology, Warsaw University of Technology) and one software development company (Nokia Siemens Networks).

It would be interesting to check whether there are any interesting trends with regard to the contribution of Polish researchers in last years. A subsequent analysis presented in Figure 1.2 shows how selected European countries contributed to the analyzed set of journals in successive years. We cannot see any specific trend with regard to Polish contribution in the last five years.

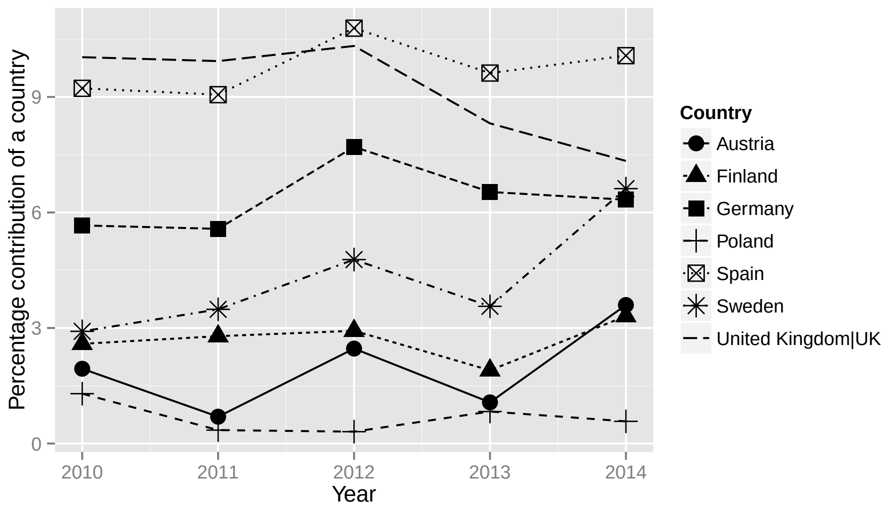


Figure 1.2. Percentage contribution of European countries to leading software engineering journals.

1.2. Identification of events in use cases

1.2.1. Research context

There are many quality attributes of requirements specifications, one of them is completeness. If one considers use cases for description of functional requirements, it is important to include complete list of events which may interrupt main scenarios. Missing events can lead to higher project costs and overrunning schedule. Therefore, a question arises: *what is the effective and efficient method to identify events in use cases?* No specific method, aimed at identification of events in use cases, had been found, hence, as the first step, method based on HAZOP approach has been proposed and evaluated in comparison to the ad hoc approach [JNOG13b]. As the second step, automatic method of events identification has been proposed and evaluated.

1.2.2. Research objectives

The goal of this study was to propose methods aimed at identification of events in use cases. Moreover, these methods have been evaluated from the stand point of accuracy and speed.

1.2.3. Research methods

HAZOP method has been used as a fundament for the proposed H4U method, which is aimed at identification of events in use cases. H4U uses the notion of primary and secondary keywords in the process of analysis of use cases. In order to evaluate the proposed approach, two controlled experiments have been designed and conducted. In both experiments the H4U method has been compared to the ad hoc approach. Participants of the first experiment included 18 students and in the second experiment 64 IT professionals were involved. In both experiments, the accuracy and speed of the two approaches have been measured and evaluated. Moreover, an automatic method of events identification has been proposed. In order to elaborate this method, 160 use cases from software projects have been analyzed. This analysis let to naming 14 abstract event types and two inference rules. The automatic method has been evaluated from the point of view of speed and accuracy. Moreover, linguistic quality of the automatically identified events has been assessed in an experiment based on the assumptions of Turing-test. Benchmark use-case-based requirements specification was used in the evaluations of ad hoc approach, H4U method and automatic method.

1.2.4. Research results

In the first place, H4U method has been evaluated with comparison to the ad hoc approach. The first experiment (with students) showed that H4U method allows to achieve more accurate results. However, the participants who used the H4U method were slower in the analysis of use cases than the participants who used ad hoc approach. The second experiment (with IT professionals) confirmed these results. The results from both experiments showed that the accuracy of events identification ranged from 0.15 to 0.26. Experiment concerning the proposed automatic method of events identification showed that this method can achieve accuracy at the level of 0.8, which is better than manual approaches. Moreover, automatic method is faster than manual methods, i.e., it is able to analyze 10.8 steps per minute, while participant of the experiments were able to analyze on average 2.5 steps per minute with ad hoc approach and 0.5 steps per minute with H4U method. In terms of linguistic quality of the automatically identified events, it can be concluded that the understandability of event descriptions generated by computer was not worse than understandability of event descriptions written by humans.

1.2.5. Conclusions

The proposed H4U method, aimed at identification of use-case events, provides effective alternative to the ad hoc approach in terms of accuracy of event identification. The accuracy and speed of identification of events can be further improved by using the proposed automatic method.

1.3. Solving the invariability problem in OCL

1.3.1. Research context

There exist various methods and languages for the specification and modeling of object-oriented systems. Contracts are the prevailing way of specifying systems from the caller point of view (see [Mey88]). The Unified Modeling Language (UML) [OMG11] is often used in combination with the Object Constraint Language (OCL) [OMG12], a high-level language for a contractual specification of object-oriented systems. In OCL, one can express invariants and operations' pre- and post-conditions.

The specification of invariable system parts is a well known problem. Usually when a change of a large system happens, only its small part is modified and the rest remains unchanged. In case of complex systems one needs a means for avoiding extensive specification of those invariable parts. This is the so called *frame problem*. In case of object-oriented systems, one has to specify what happens with all objects' attributes and associations. However, without a proper means the resulting specification can be, and often is, very excessive. For a number of years this problem remained unsolved for OCL.

The frame problem

In general there exist four approaches to the frame problem: minimal-change approach, implicit specification and frame formulas. The minimal-change approach requires that the set of changed system parts is minimal, i.e., the change is in accordance with the specification in the usual sense and moreover the number of changed parts cannot be smaller. A serious disadvantage of this approach is that it is very hard to figure out such minimal sets, and the minimality proof can be complex and non-standard. Thus it is not useful in practice, and specially when tool support is needed

An implicit approach to invariability was used in case of OCL, however it dates back to Hoare logic. In this logic all variables which are not mentioned in the so called Hoare triple are assumed to be unchanged. The idea of the approach is similar: all system parts not mentioned in a specification must not change. This approach allows to write simple specifications and does not require any special means. However, it does not work well for contractual specifications because an operation execution can have very complex side-effects. This approach also heavily depends on the actual form of the specification and for specifications equivalent in the classical logical sense it may result in different variable parts.

The frame formulas are used in artificial intelligence (cf. [Sch90]). The idea is to specify modification of attributes using axiom schemata. This approach requires however proper means to make the specifications compact. In case of Java Modeling Language (JML, see [DM05]) and also Spec#, explicit invariability clauses are used for a compact specification of invariable properties. Invariability constraints can be checked at the compile-time. Thus, it is not possible to specify invariability requirements which cannot be checked statically. Moreover, these languages are much simpler than OCL.

1.3.2. Research objectives

The goal of research [Kos13c; Kos13a] was to provide specification primitives which address the shortcomings of previously existing approaches. In particular, these primitives should:

— allow one to specify the invariable part of object-oriented systems
— be language-based, not semantics-based, preferably OCL-based
— allow validity monitoring with standard OCL-tools
— ensure logical equivalence of OCL-specifications in the standard sense
— allow for the application of standard proof techniques

1.3.3. Research results

A state of an object-oriented system can be understood as a graph with labeled nodes and labeled edges. The nodes correspond to objects, their labels to their classes. The edges correspond to links between objects; the corresponding labels to class attributes and associations among them. A state change of an oo-system corresponds to a state change of such a labeled graph. Such a change concerns object creation and deletion, and also link modification. We let the object creation and deletion be governed by the OCL-specification. However we add new primitives to identify sets of those links which can be removed or replaced by new ones. Thus, a system change, object removal and creation as well as link modification can happen as long as the basic OCL-specification is satisfied and the links are modified only when they are specified by the primitives.

An OCL-specification of an operation has basically three parts. The first part of the specification declares the context, i.e., the signature of specified method and the class it belongs to. The second part specifies its pre-condition, i.e., the condition which has to be satisfied before the operation is executed. The third part is the post-condition, i.e., a condition which must be valid after the operation's execution. We add to operations' post-conditions invariability clauses of the form:

in p modifies t_1::a_1, ..., t_n::a_n

Clause `p` defines a set of classes; in general it can be a metamodel-based view definition [Kos13c]. Term t_i is an OCL-term defining a set of objects of a class C_i and a_i is an attribute or an association-end of this class. For objects defined by term t_i attribute/association a_i can be changed. Attribute a_i has to belong to classes defined by `p`. We do not specify in this clause, what happens with attributes not included in `p`.

As an example, consider a bank account with attribute `balance` storing information on the actual balance of a bank account and method `credit`. The way this method operates can be specified in the extended OCL in the following way:

```
context BankAccount::credit(amount : Real)
post : self.balance = self.balance@pre + amount
in BankAccount modifes : self::balance
```

The primitive `@pre` can occur only in post-conditions. It delivers the value of attribute `balance` in the pre-state, i.e., in the state before the method execution. When it does not occur as a postfix of an attribute, then the value of attribute is computed in the post-state. Operation `credit` increases attribute `balance` by adding `amount`. The invariability clause says that `credit` modifies only the attribute `balance` of objects from class `BankAccount`.

1.3.4. Conclusions

Specification of invariability was a real problem in case of OCL-specifications. The problem with designing invariability primitives was the descriptive power of OCL and the plenitude of constructs facilitating specification writing. As a result of the presented research, a solution of this problem was proposed, which is simple in form and has natural semantics. It allows one for compact specifications of invariable system parts in a compact and precise way. The semantics is defined in terms of standard OCL; this allows for the application of standard OCL-models, techniques and tools. However, there are still issues to be addressed. For example, we need to define primitives for associations with multiple ends.

1.4. Predicting the Flow of Defect Correction Effort

1.4.1. Research context

Extensive literature on defect prediction usually deals with predicting number of defects or defect proneness of a software component. While such information is useful in many contexts it does not answer the question that is more important from the resource management perspective, i.e., *how much effort will be required to correct these defects?* The described study investigated possibility of predicting correction effort instead of raw defect count. [SRGR13b]

The environment for this study was the automotive company where software is developed according to the industrial standard V-model [Ind92] with four successive phases: requirements (RE), design (DE), implementation (IM), integration and testing (I&T). An earlier study [SRGR11] had confirmed that defect correction effort (DCE) depends on the phases where a defect was inserted and detected. Specifically, defects inserted in early phase, but detected in later, need more effort for their correction than if they are inserted and detected in the same phase. This flow of defect correction effort between phases makes the main rationale for the proposed predictive model.

1.4.2. Research objectives

The main goal of this study was to develop a model that could predict the defect correction effort at various development phases. This model, called a Defect Cost Flow Model (DCFM) reflects a V-model of a software development lifecycle – a real engineering process for developing embedded applications in the automotive industry. With this model it was possible to **optimize the amount of quality assurance** (QA) activities in different phases to minimize the total project effort.

1.4.3. Research methods

Technically, the DCFM is a Bayesian Network (BN). Among various reasons for choosing a Bayesian Network as a formal representation of DCFM the most important were:
- Model structure reflects cause-effect relationships for better understanding and fit to reality.
- BNs may incorporate expert knowledge combined with empirical data.
- They enable performing various types of analyses using rigorous probability calculus focused on decision support.

The research process involved the following main phases:
1. Problem definition using the Goal-Question-Metric approach.
2. Data gathering and analysis – using the internal *change and defect management system* as main source. The second data source was expert knowledge from researchers, developers and managers supporting this study. Exiting literature in the field served as the third data source.
3. Model creation and enhancement covered building initial version of the model as well as its multiple enhancements. Each version contained new elements (i.e. variables) and the whole model was calibrated using the data obtained in the previous phase. The model was created in an iterative process for easier validation and access to the working (partial) version at each time.
4. Model validation covered general model behavior, practical usefulness, detailed model behavior in numerous scenarios with different input data, sensitivity analysis, and in the possibility of adjusting and calibrating the core of DCFM.

1.4.4. Research results

The main result was the Defect Cost Flow model. Its structure is too large for display and discuss in a single figure. Thus, Figure 1.3 illustrates the core structure of the model while Figure 1.4 presents some details for the design phase. The defect correction effort flows from the phase where defects are inserted until they are detected and fixed.

Specifically, some defects inserted in the requirements phase are also fixed there. But since the review process is imperfect some defects are left and thus flow to the next phase (design). Correcting these defects in this phase requires more effort (4-5 times) as reflected by the *effort multiplier* (Figure 1.4 left). With higher *level of QA activity* more defects could be detected in the design phase. But still, some would be left and detected in later phases.

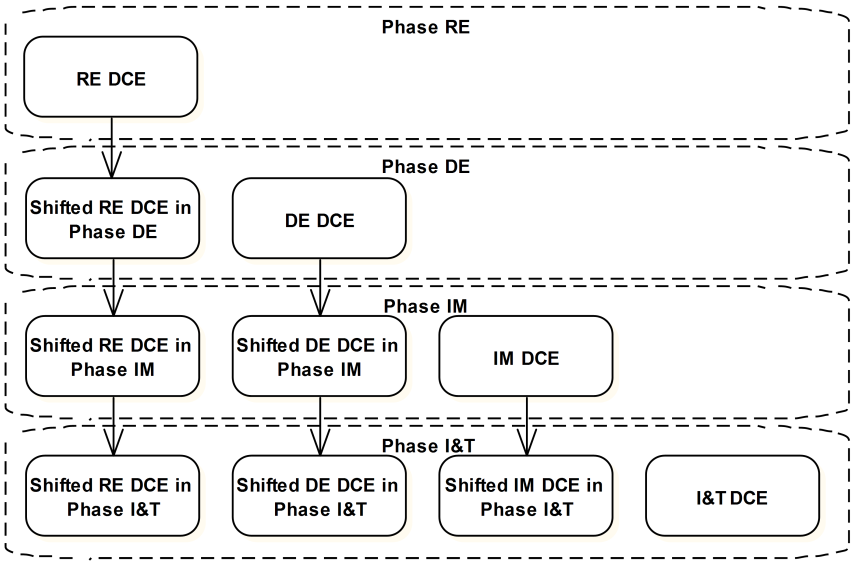

Phase RE
RE DCE
Phase DE
Shifted RE DCE in Phase DE
DE DCE
Phase IM
Shifted RE DCE in Phase IM
Shifted DE DCE in Phase IM
IM DCE
Phase I&T
Shifted RE DCE in Phase I&T
Shifted DE DCE in Phase I&T
Shifted IM DCE in Phase I&T
I&T DCE

Figure 1.3. Schematic of DCFM [SRGR13b].

In the design phase also new defects are inserted. They would need to be detected and fixed – partially in the QA activity, and partially in the next phases as the defect correction effort (Figure 1.4 right).

The model incorporates various empirical data, e.g.:

— The probabilities for inserting defects slowly decreases in the first three phases and drops down rapidly in the I&T phase.
— Different levels for sufficiency of QA effort are defined as a percentage of the core development effort.
— The QA activities are the most efficient in the RE and DE phases. In IM and I&T phases they are significantly less efficient in detecting defects originating from earlier phases.

1.4.5. Conclusions

Model validation confirms that the model provides sensible predictions consistent with gathered empirical data and known literature in software engineering field. What is especially important is that this model has been applied in a real in-

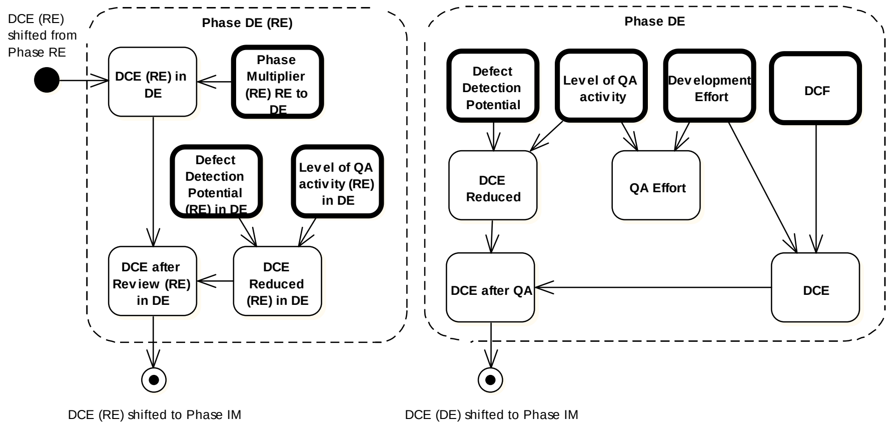

DCE (RE) shifted from Phase RE
Phase DE (RE)
DCE (RE) in DE
Phase Multiplier (RE) RE to DE
Defect Detection Potential (RE) in DE
Level of QA activity (RE) in DE
DCE after Review (RE) in DE
DCE Reduced (RE) in DE
DCE (RE) shifted to Phase IM
Phase DE
Defect Detection Potential
Level of QA activity
Development Effort
DCF
DCE Reduced
QA Effort
DCE after QA
DCE
DCE (DE) shifted to Phase IM

Figure 1.4. Model structure for design phase (DE) [SRGR13b].

dustrial process. It demonstrates high potential in finding the appropriate amount of review effort for specific development phases to minimize the overall costs. Thus, the model may be used in the industry for decision support. By extending and calibrating it can be tailored to meet the needs of specific development environment.

1.5. Model of a maturity capsule in software project management

1.6. Research context

Research conducted at the Center for Advanced Studies on Campus (CAS) focuses on issues of software project management and on finding solutions to improve management and development processes. The development methods used in project management and in the development environment prove inadequate to the problems of contemporary IT projects. It is therefore proposed that in project management the development and management processes should be monitored with the use of the innovative maturity capsule developed at CAS.

1.6.1. Research objectives

The main objective of the study was to define and apply in practice the maturity capsule in IT projects. It was assumed that the concept of the maturity capsule is to be understood as a set of maturity ratings of the supplier, client and project (estimated through the scalar negentropy of the project) [KO14a]. To define the maturity capsule, it is necessary to establish the measurements of the maturity of the client and the supplier and the project negentropy. The knowledge resulting from the COBIT (Control Objectives for Information and Related Technology) and ITIL (Information Technology Infrastructure Library) standards is important for the initial processing of the project data, which aims at evaluating the maturity of the supplier and client organizations in question. The TOGAF standard is used mainly to evaluate a specific indicator, measuring the degree of global maturity of a project, called negentropy. The applicability of the model is verified in a number of environments, mainly in IT projects and in the organizations carrying out such projects [KO14b].

1.6.2. Research methods

The description of the project management processes and information technologies was based on a formal, discrete - time-linear dynamic description expanded with the essential nonlinear mechanisms in the form of a fuzzy - rule-based system (with the use of a linguistic estimation developed on the basis of answers given by experts to sets of questions in interview questionnaires). These descriptions were used to develop a useful model based on a sequence of the following three steps [SO14]:

1. the fuzzy modeling philosophy, based on the formation of the membership function, is an appropriate foundation of the universality of the maturity capsule.
2. the number of times of use is a relevant criterion for assessing the quality of the developed model.
3. the tuning of the model becomes possible through the identification of its parameters and variables (it is based on linguistic evaluations resulting from competency questions).

1.6.3. Research results

The conducted studies demonstrated how the maturity capsule can be used by those managing projects, by development teams and by customer teams in order to support the processes inside the project in terms of monitoring and predicting its development. Four levels of verification of the maturity capsule were proposed [Orł14]. On the first one, the usefulness of the capsule for the managers of IT projects was evaluated. On the second level, the support for processes which ensure corporate governance was referred to, as well as the use of project negentropy in supporting the management processes of a company. The third level focused on the linguistic evaluation of supplier organization maturity in predicting its evolution. While, on the fourth level of ’control of the level of the client organization and the processes of its change, a linguistic description was used to support this evaluation.

1.6.4. Conclusions

In previous studies on project management processes, neither the analysis of the state nor the maturity of the project was as comprehensive as this one. Both elements have been included in the maturity capsule to predict and optimize information technologies in managing information technology projects. The presented analysis of the maturity capsule, the possibility to progress in terms of maturity, and the monitoring of the level of management all allow for predicting technologies to support the desired changes in the maturity capsule. In this sense, the developed solution provides an innovative perspective on the management processes of technologies and IT projects, involving the aggregation of knowledge about the maturity of the entities in the capsule (client, project and supplier) and the decomposition of information technologies into services and IT functionalities. The solution described in this work, regarding a comprehensive evaluation of a project and involving the use of the maturity capsule, requires indicating how frequent this evaluation is and analyzing its applicability for the organization/teams of the client and the supplier, which change dynamically during the project. In such cases, the standard use of evaluation questionnaires may be inadequate. A better solution would be to develop and apply a system in which specialized agents make the assessment. The purpose of such a system would be to evaluate the environment in which the IT project is carried out.

1.7. Conclusions

The contribution of Polish researchers to the software engineering research field is limited. The percentage of research papers in the analyzed set of leading software engineering journals and the time period (2010 – 2014 Sep 14) was about 0.6%. Fortunately, there are some valuable achievements of Polish researchers which we tried to present briefly in this chapter.

It is also worth mentioning that we did not analyzed which research institutions in Poland contribute the most as sometimes changing an affiliation of one or two researchers would influence the results to a large extent.

References

[Gla94] R. L. Glass. “An Assessment of Systems and Software Engineering Scholars and Institutions”. In: *Journal of Systems and Software* 27.1 (Oct. 1994), pp. 63–67. ISSN: 0164-1212. DOI: *10.1016/0164-1212(94)90115-5*. URL: *http://dx.doi.org/10.1016/0164-1212(94)90115-5*.

[WTGBC09] W. E. Wong, T. H. Tse, R. L. Glass, V. R. Basili, and T. Y. Chen. “Controversy Corner: An Assessment of Systems and Software Engineering Scholars and Institutions (2002-2006)”. In: *Journal of Systems and Software* 82.8 (Aug. 2009), pp. 1370–1373. ISSN: 0164-1212. DOI: *10.1016/j.jss.2009.06.018*. URL: *http://dx.doi.org/10.1016/j.jss.2009.06.018*.

[WTGBC11] W. E. Wong, T. Tse, R. L. Glass, V. R. Basili, and T. Chen. “An assessment of systems and software engineering scholars and institutions (2003–2007 and 2004–2008)”. In: *Journal of Systems and Software* 84.1 (2011). Information Networking and Software Services, pp. 162 –168. ISSN: 0164-1212. DOI: *http://dx.doi.org/10.1016/j.jss.2010.09.036*. URL: *http://www.sciencedirect.com/science/article/pii/S0164121210002682*.

[Woh09] C. Wohlin. “An Analysis of the Most Cited Articles in Software Engineering Journals - 2002”. In: *Information and Software Technology* 51.1 (Jan. 2009), pp. 2–6. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2008.09.012*. URL: *http://dx.doi.org/10.1016/j.infsof.2008.09.012*.

[Mad10] L. Madeyski. “The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment”. In: *Information and Software Technology* 52.2 (2010), pp. 169–184. DOI: *10.1016/j.infsof.2009.08.007*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.08.007*.

[ONK11] M. Ochodek, J. Nawrocki, and K. Kwarciak. “Simplifying Effort Estimation Based on Use Case Points”. In: *Information and Software Technology* 53.3 (Mar. 2011), pp. 200–213. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2010.10.005*. URL: *http://dx.doi.org/10.1016/j.infsof.2010.10.005*.

[MR10] L. Madeyski and N. Radyk. “Judy – A Mutation Testing Tool for Java”. In: *IET Software Journal (formerly IEE Proceedings*

Software) 4.1 (2010). Draft: *http://madeyski.e-informatyka.pl/download/Madeyski10b.pdf*, pp. 32–42. DOI: *10.1049/iet-sen.2008.0038*. URL: *http://dx.doi.org/10.1049/iet-sen.2008.0038*.

[JZ10a] A. Janik and K. Zielinski. “AAOP-based Dynamically Reconfigurable Monitoring System”. In: *Information and Software Technology* 52.4 (Apr. 2010), pp. 380–396. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2009.10.006*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.10.006*.

[NK12] G. J. Nalepa and K. Kluza. “UML REPRESENTATION FOR RULE-BASED APPLICATION MODELS WITH XTT2-BASED BUSINESS RULES”. In: *International Journal of Software Engineering and Knowledge Engineering* 22.04 (2012), pp. 485–524. DOI: *10.1142/S021819401250012X*. URL: *http://www.worldscientific.com/doi/abs/10.1142/S021819401250012X*.

[Mił10] M. Miłkowski. “Developing an Open-source, Rule-based Proofreading Tool”. In: *Software: Practice and Experience* 40.7 (June 2010), pp. 543–566. ISSN: 0038-0644. DOI: *10.1002/spe.v40:7*. URL: *http://dx.doi.org/10.1002/spe.v40:7*.

[JZ10b] A. Janik and K. Zielinski. “Adaptability Mechanisms for Autonomic System Implementation with AAOP”. In: *Software: Practice and Experience* 40.3 (Mar. 2010), pp. 209–223. ISSN: 0038-0644. DOI: *10.1002/spe.v40:3*. URL: *http://dx.doi.org/10.1002/spe.v40:3*.

[BBZ10] P. Bachara, K. Blachnicki, and K. Zielinski. “Framework for Application Management with Dynamic Aspects J-EARS Case Study”. In: *Information and Software Technology* 52.1 (Jan. 2010), pp. 67–78. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2009.06.003*. URL: *http://dx.doi.org/10.1016/j.infsof.2009.06.003*.

[Flo+10] J. Floch et al. “A Comprehensive Engineering Framework for Guaranteeing Component Compatibility”. In: *Journal of Systems and Software* 83.10 (Oct. 2010), pp. 1759–1779. ISSN: 0164-1212. DOI: *10.1016/j.jss.2010.04.075*. URL: *http://dx.doi.org/10.1016/j.jss.2010.04.075*.

[Deo10] S. Deorowicz. “Solving Longest Common Subsequence and Related Problems on Graphical Processing Units”. In: *Software: Practice and Experience* 40.8 (2010), pp. 673–700. ISSN: 0038-0644.

DOI: *10.1002/spe.v40:8*. URL: *http://dx.doi.org/10.1002/spe.v40:8*.

[ZK13] A. Zalewski and S. Kijas. "Beyond ATAM: Early Architecture Evaluation Method for
Large-scale Distributed Systems". In: *Journal of Systems and Software* 86.3 (Mar. 2013), pp. 683–697. ISSN: 0164-1212. DOI: *10.1016/j.jss.2012.10.923*. URL: *http://dx.doi.org/10.1016/j.jss.2012.10.923*.

[ŁM12] K. Łukasiewicz and J. Miler. "Improving agility and discipline of software development with the Scrum and CMMI". In: *Software, IET* 6.5 (2012), pp. 416–422. ISSN: 1751-8806. DOI: *10.1049/iet-sen.2011.0193*.

[JKP12] M. Janicki, M. Katara, and T. Pääkkönen. "Obstacles and Opportunities in Deploying
Model-based GUI Testing of Mobile Software: A Survey". In: *Software Testing, Verification and Reliability* 22.5 (Aug. 2012), pp. 313–341. ISSN: 0960-0833. DOI: *10.1002/stvr.460*. URL: *http://dx.doi.org/10.1002/stvr.460*.

[OAJN11] M. Ochodek, B. Alchimowicz, J. Jurkiewicz, and J. Nawrocki. "Improving the Reliability of Transaction Identification in Use Cases". In: *Information and Software Technology* 53.8 (2011), pp. 885–897. ISSN: 0950-5849. DOI: *10.1016/j.infsof.2011.02.004*. URL: *http://dx.doi.org/10.1016/j.infsof.2011.02.004*.

[Hof11] R. Hofman. "Behavioral Economics in Software Quality Engineering". In: *Empirical Software Engineering* 16.2 (Apr. 2011), pp. 278–293. ISSN: 1382-3256. DOI: *10.1007/s10664-010-9140-x*. URL: *http://dx.doi.org/10.1007/s10664-010-9140-x*.

[JNOG13a] J. Jurkiewicz, J. Nawrocki, M. Ochodek, and T. Głowacki. "HAZOP based identification of events in use cases". English. In: *Empirical Software Engineering* (2013), pp. 1–28. ISSN: 1382-3256. DOI: *10.1007/s10664-013-9277-5*. URL: *http://dx.doi.org/10.1007/s10664-013-9277-5*.

[RDDGM12] A. Riel, A. Draghici, G. Draghici, D. Grajewski, and R. Messnarz. "Process and product innovation needs integrated engineering collaboration skills". In: *Journal of Software: Evolution and Process* 24.5 (2012), pp. 551–560. ISSN: 2047-7481. DOI: *10.1002/smr.497*. URL: *http://dx.doi.org/10.1002/smr.497*.

[JS14] P. Janczarek and J. Sosnowski. “Investigating software testing and maintenance reports: Case study”. In: *Information and Software Technology* 0 (2014), pp. –. ISSN: 0950-5849. DOI: *http://dx.doi.org/10.1016/j.infsof.2014.06.015*. URL: *http://www.sciencedirect.com/science/article/pii/S0950584914001542*.

[MJ14] L. Madeyski and M. Jureczko. “Which Process Metrics Can Significantly Improve Defect Prediction Models? An Empirical Study”. In: *Software Quality Journal* (accepted) (2014). DOI: *10.1007/s11219-014-9241-7*. URL: *http://dx.doi.org/10.1007/s11219-014-9241-7*.

[MOTJ14] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. “Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation”. In: *IEEE Transactions on Software Engineering* 40.1 (2014), pp. 23–42. ISSN: 0098-5589. DOI: *10.1109/TSE.2013.44*. URL: *http://dx.doi.org/10.1109/TSE.2013.44*.

[Sob14] J. Sobecki. “Comparison of Selected Swarm Intelligence Algorithms in Student Courses Recommendation Application”. In: *International Journal of Software Engineering and Knowledge Engineering* 24.01 (2014), pp. 91–109.

[CC13] B. Czarnacka-Chrobot. “RATIONALIZATION OF BUSINESS SOFTWARE SYSTEMS DEVELOPMENT AND ENHANCEMENT PROJECTS INVESTMENT DECISIONS ON THE BASIS OF FUNCTIONAL SIZE MEASUREMENT”. In: *International Journal of Software Engineering and Knowledge Engineering* 23.06 (2013), pp. 839–868. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013500228*.

[SRGR13a] T. Schulz, Ł. Radliński, T. Gorges, and W. Rosenstiel. “Predicting the Flow of Defect Correction Effort using a Bayesian Network Model”. English. In: *Empirical Software Engineering* 18.3 (2013), pp. 435–477. ISSN: 1382-3256. DOI: *10.1007/s10664-011-9175-7*. URL: *http://dx.doi.org/10.1007/s10664-011-9175-7*.

[Kos13b] P. Kosiuczenko. “Specification of Invariability in OCL”. In: *Software and Systems Modeling* 12.2 (May 2013), pp. 415–434. ISSN: 1619-1366. DOI: *10.1007/s10270-011-0215-y*. URL: *http://dx.doi.org/10.1007/s10270-011-0215-y*.

[Ped13] W. Pedrycz. "KNOWLEDGE MANAGEMENT AND SEMANTIC MODELING: A ROLE OF INFORMATION GRANULARITY". In: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 5–11. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013400019*.

[JKN13] J. J. Jung, R. P. Katarzyniak, and N. T. Nguyen. "GUEST EDITORS; INTRODUCTION". In: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 1–3. DOI: *10.1142/S0218194013020014*.

[KP13] R. P. Katarzyniak and G. Popek. "INTEGRATION OF MODAL AND FUZZY METHODS OF KNOWLEDGE REPRESENTATION IN ARTIFICIAL AGENTS". In: *International Journal of Software Engineering and Knowledge Engineering* 23.01 (2013), pp. 13–29. DOI: *10.1142/S0218194013400020*. URL: *http://www.worldscientific.com/doi/abs/10.1142/S0218194013400020*.

[PŻZ13] M. Psiuk, D. Żmuda, and K. Zielinski. "Distributed OSGi Built over Message-oriented Middleware". In: *Software: Practice and Experience* 43.1 (Jan. 2013), pp. 1–31. ISSN: 0038-0644. DOI: *10.1002/spe.1148*. URL: *http://dx.doi.org/10.1002/spe.1148*.

[JNOG13b] J. Jurkiewicz, J. Nawrocki, M. Ochodek, and T. Głowacki. "HAZOP based identification of events in use cases". In: *Empirical Software Engineering* (2013), pp. 1–28.

[Mey88] B. Meyer. *Object-oriented software construction*. Prentice Hall New York, 1988.

[OMG11] OMG. *Unified Modeling Language, Spec. ver. 2.4.1*. 2011.

[OMG12] OMG. *Object Constraint Language, Spec. ver. 2.3.1*. 2012.

[Sch90] L. Schubert. "Monotonic solution of the frame problem in the situation calculus". In: *Knowledge representation and defeasible reasoning*. Springer, 1990, pp. 23–67.

[DM05] A. Darvas and P. Müller. "Reasoning about method calls in JML specifications". In: *Proceedings of FTfJP'05*. 2005.

[Kos13c] P. Kosiuczenko. "Specification of invariability in OCL, Specifying invariable system parts and views". In: *Software & Systems Modeling* 12.2 (2013), pp. 415–434.

[Kos13a] P. Kosiuczenko. "On the Validation of Invariants at Runtime". In: *Fundamenta Informaticae* 125.2 (2013), pp. 183–222.

[SRGR13b] T. Schulz, Ł. Radliński, T. Gorges, and W. Rosenstiel. "Predicting the Flow of Defect Correction Effort using a Bayesian Network Model". English. In: *Empirical Software Engineering* 18.3 (2013), pp. 435–477. ISSN: 1382-3256. DOI: *10.1007/s10664-011-9175-7*. URL: *http://dx.doi.org/10.1007/s10664-011-9175-7*.

[Ind92] Industrieanlagen-Betriebsgesellschaft (IABG). *The V-Model — General Directive 250. Software Development Standard for the German Federal Armed Forces*. Ottobrunn, Germany, 1992.

[SRGR11] T. Schulz, Ł. Radliński, T. Gorges, and W. Rosenstiel. "Software Process Model using Dynamic Bayesian Networks". In: *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*. Ed. by M. Ramachandran. Hershey: Information Science Reference, 2011, pp. 289–310. ISBN: 978-1-60960-509-4. DOI: *10.4018/978-1-60960-509-4.ch016*.

[KO14a] Z. Kowalczuk and C. Orłowski. *Advanced Modeling of Management Processes in Information Technology*. Springer, 2014, pp. 1–203. ISBN: 978-3-642-40876-2.

[KO14b] Z. Kowalczuk and C. Orłowski. "Model of a Maturity Capsule in Managing IT Projects". In: *Cybernetics and Systems* 45.2 (2014), pp. 123–135.

[SO14] E. Szczerbicki and C. Orłowski. "Guest Editorial: Designing and Developing Smart Cognitive Systems: Implementation Lessons from the Real World". In: *Cybernetics and Systems* 45.2 (2014), pp. 89–91.

[Orł14] C. Orłowski. "Rule-based model for selecting integration technologies for Smart Cities systems". In: *Cybernetics and Systems* 45.2 (2014), pp. 136–145.

Chapter 2

Working with Agile in a Distributed Environment

This chapter describes the approach of Capgemini to agile projects in a distributed environment. It presents key methods and tools to minimize the geographical and mental distance within distributed teams, for example: communication between all team members through excellent technical infrastructure – voice-, video- and desktop sharing tools, using well known best communication practices as well as exchange programs, where people from each location travel and visit each other on a regular basis.

There is also a discussion of organizational models for distributed agile teams as well as new project roles enabling better communication and knowledge sharing.

The system architecture and the role of an architect in agile teams is affected by application of the best practices and techniques, including, but not limited to, less detailed upfront design, architects working hands-on, test driven development, clean code rules and domain driven design.

The Capgemini Agile! Nearshore calls for transparency in projects, by sharing the Product and Sprint Backlogs in digital form, common knowledge-sharing space and shared project infrastructure.

Agile methodologies are gaining more and more recognition in IT projects. According to 2011 CHAOS [CHA11, p. 25] report from the Standish Group Software, applications developed through the agile process have three times the success rate of the traditional waterfall method and a much lower percentage of time and cost overruns. However agile processes are not a 100 percent recipe for success, especially in case of distributed projects. The decision to distribute team members is made for a reason - to save money, to recruit in multiple locations, to gain experts or because of client offices location [Coh09, pp. 386-387].

There are several factors that determine the success of a project. One of the most important aspects is the team [CH01]. The feeling of being part of a team is not to be underestimated. The team creates short-and long-term values of the project. The team is highly motivated to develop their skills. The team understands how to effectively achieve the objectives of the project. The ability to manage a distributed team is a big challenge.

Based on Capgemini Rightshore® [1] concept and previous agile experiences, we have found the following success factors for distributed agile projects:

1. Proper staffing.
2. Proper organization structure.
3. Communication excellence and proper tooling.
4. Right software architecture and technology.

2.1. Architecture in agile engagements

Architecture and the role of an architect are not actually referred to in the agile approach. However, based on our experience, it changes the way of creating architecture and redefines the role of an architect compared to more traditional approaches. This chapter describes the experience in the form of best practices and techniques.

A detailed upfront design is very common in the waterfall model. Contrary to it, the agile approach assumes constantly changing requirements; consequently the architecture created at the very beginning may become outdated. To address this issue Capgemini Agile! Nearshore recommends to start a project with a less detailed upfront design, which gives the team more flexibility when requirements force architectural changes. In some areas requirements alone drive architectural maturity. This has another consequence: not only architects but also developers may create architecture as this may happen during development. This, in turn, raises a question whether an architect is still needed in agile engagements. Our experience shows that in middle- and large-scale agile projects we still need architects, although their role differs from the commonly met traditional approaches.

Beside the best practices mentioned above, we have gained practical experience in applying some design and implementation techniques which are crucial when it comes to agile engagements. The Domain Driven Design [Eva04] makes the domain and its jargon understandable within a team. Since the domain model is independent of technical solutions and not under their influence, it remains up-to-date all the time.

The Test Driven Development [Bec02] makes the code ready for changes as there are tests enabling regression testing. The quality of these tests help verify the developers' understanding of the requirements. The rules of the Clean

[1] Rightshore® is Capgemini's global distributed delivery approach where we provide the right resources, the right location at the right time to clients by leveraging an industrialized approach. We do this through a global network of centers: on shore, nearshore and offshore. In this article, we shall focus on nearshore projects.

Code [Mar08] make the code readable and - consequently - easier to maintain under constant changes.

2.2. Minimize distance with Agile - Agile!Nearshoring

What to do when adopting Agile in a team? The first thing is to make sure a team has been forged. The teamwork is essential in this context. The team members have to recognize everybody's strengths and weekneses. If they can work that out between them then they know how they can help each other. If they are not willing to help each other, their chance of success is really slim. So make them all realize what their roles can be, what they can do to make sure they work successfully together as a team and then to find out that working together gives far more rewarding experience.

We decided to use the classical Scrum [SS13] approach as a basis. The Scrum is most common industry agile standard, a flexible and adaptive framework. Using Scrum processes and agile ceremonies [WSG10, pp. 1-17], nothing changes because of Nearshoring. However, due to the specific nature of working in distributed projects, we have paid particular attention to the team members, the organization of the team and the tools that facilitate communication.

2.2.1. The team structure

The right team mix is very important for good cooperation of the Scrum Team. Preference is given to teams of developers who have previously worked together. Team members selected for the agile project must have better communication and language skills than the average in other projects. Team members in total should have broad technical and business domain knowledge to be able to address different issues. At least one person should have a good ability to find experts or expertise in a specific area, using internal and external resources. Every team member must be aware of their responsibility for the project goals and results. Given that one of the aspects determining the success of the project is domain knowledge, we have proposed the introduction of a new role - Product Owner Proxy (PP).

Product Owner Proxy. In principle, this role is filled by a business architect fully experienced in the domain. This role is to support the client and the team only in the specific area of knowledge; other tasks of the Product Owner, such as product backlog prioritization, release planning, and whether to accept or reject work results, are not supported by the Product Owner Proxy role. Usually, the

Product Owner Proxy also has technical skills. It is important, because especially in the early project stages, they work as a bridge between developers, who tend to use technical language, and the customer who is rather business oriented.

There is also the aspect of Product Owner's availability. The person assigned to this function is often rarely available for the team, and in particular, cannot travel regularly between locations.

Team organization. Another important element is the organization of a team - especially at the beginning of the project. We want to avoid a situation, where despite the right people and tools, the projects are not able to deliver business value, due to organizational problems. We present two organizational patterns, which are used as a base for new projects. These models are designed to facilitate the start of the project, particularly if the team does not have experience in agile methodologies. Another aspect is the support of organization, especially when the client has no previous experience in developing systems based on the Scrum framework. These models, depending on the needs of the client or the team, may be modified or scaled in subsequent sprints [Coh09, pp. 325-350].

Model 1: The entire Development Team and the Scrum Master in a remote location The main assumption of this model of organization is to locate the entire development team in one location, as shown in figure 2.1. The main idea and the team structure are described below.

— The entire Development Team and the Scrum Master in a remote location.
— Product Owner comes from the Customer's organization.
— Product Owner Proxy is onsite (or alternatively in a remote location).
— Scrum Master is in a remote location.
— Development Team is in a remote location.

The whole Development Team is seated together. They can organize their cooperation more efficiently. What is extremely important from the point of view of agile methodologies, working in one room or in rooms close to each other (e.g., on the same floor) significantly affects the flow of information and their quality. Also, there is no need for a distributed developing environment. This is particularly important for teams using the Pair Programming technique [Bec00], which is derived from the XP methodology [Bec99].

Only one person (Product Owner Proxy from Capgemini), or alternatively two persons (both Product Owners) travel frequently.

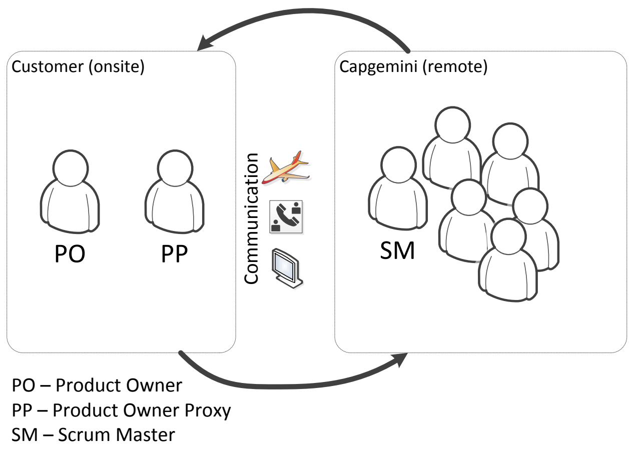


Figure 2.1. Model 1: The entire Development Team and the Scrum Master in a remote location.

Although in principle, the team is working remotely, we advice that the first 2 Sprints should be performed onsite to better understand the customer's needs and to establish personal relationships.

There are also disadvantages of this setup. Usually the personal access to the Product Owner from customer for the team members is only occasional. Most conversations are held via telephone, which affects the quality of the information passed around. The same goes for end users, who are always a valuable source of information and data for agile teams.

The environment for acceptance tests is difficult to maintain, because there are no team members onsite who could take care of it. This problem mainly occurs when a client does not have its own specialists responsible for IT infrastructure.

Recurring onsite meetings (with customer) for demos and backlog prioritization are a necessity. For large distances, delegating some employees significantly affects the team capacity available for development tasks. Traveling is also associated with a significant financial burden on the project - especially for small and medium-sized projects.

Model 2: The Development Team is distributed In this case part of the team is on the client's side (figure 2.2). This model is recommended when the task must necessarily be carried out at the customer's premises or where part of the team (due to their place of residence) may work directly. The main idea and the team structure are described below.

— Product Owner comes from the Customer's organization.
— Product Owner Proxy onsite (or alternatively in the remote location).
— Scrum Master in the location where the major part of the team is located. Self-organizing Team can choose a local coordinator. Such person should take care of organizational and infrastructural aspects (booking video conference rooms, local databases, etc) but also help to understand and apply the agile rules. Ideally he or she should have experience in agile projects.
— Development Team onsite and in a remote location (distributed).
— The team members from different locations should build small domain teams. Separating team members from different locations should be avoided.

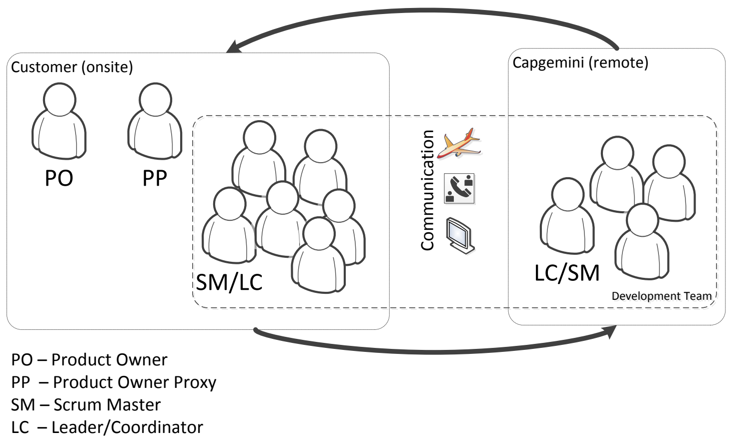


Figure 2.2. Model 2: The Development Team is distributed.

Usually the visits are bilateral, all team members have personal contact with the Product Owner from the customer.

Coordination of the distributed team is more difficult and expensive. Development environments must be maintained on both sides if remote access is not possible. Despite the fact that some of the developers work on the client side, it is worth ensuring that remote team members make customer visits regularly. This is to form relationships and mutual trust. In the event of significant communication problems, it is worth temporarily relocating 1-2 members of the remote team to work on the client side. Seeding visits and traveling ambassadors are excellent ways to pick up the local customs and preferences of a team in one location and return them to another location [Coh09, pp. 370-372].

2.2.2. Tools - the way to minimize distance

The role of tools with Agile teams and processes Tooling is important. Obviously, the process comes first, people change, culture, team etc. But often many things get large, they get complex, they get distributed. Tooling becomes essential. When it comes to communication and cooperation, people typically work on complex tasks, which makes using tools essential [AL12]. So we mainly focus on getting tools that help people collaborate.

So if you've got a large team involved in big projects, you need to be able to launch communication, not just in that small agile team (7 +/-2 people), but also when you get circa 100 people engaged at the same time. You can't have everybody physically talking to each other. You need to be able to get communication through tooling. Tooling gives everybody an insight as to what's really going on and allows everybody to understand the big picture and what actual progress the team has made. While discussing the tools in IT projects we focus on the following objectives:
— increasing the effectiveness of meetings (method of communication),
— encouraging the team to observe processes and participate in its creation.

Communication effectiveness. An important part of communication is nonverbal, and a huge challenge in communication between distributed team members is missing out on nonverbal clues [WSG10, p. 20]. Social anthropologists argued that in a normal conversation more than 65 percent [Hal90, Bor08] of social meaning is transferred through the nonverbal channel.
The notion of effective communication includes such communication that benefits each of the parties [Coc06], e.g.:
— the necessary information is obtained,
— the decisions taken are clear and understandable,
— meetings only involve the necessary people,
— the limitations of each communication channel are gradually eliminated.
In figure 2.3, you can see the relationship between the effectiveness of communication and the type of tools used.

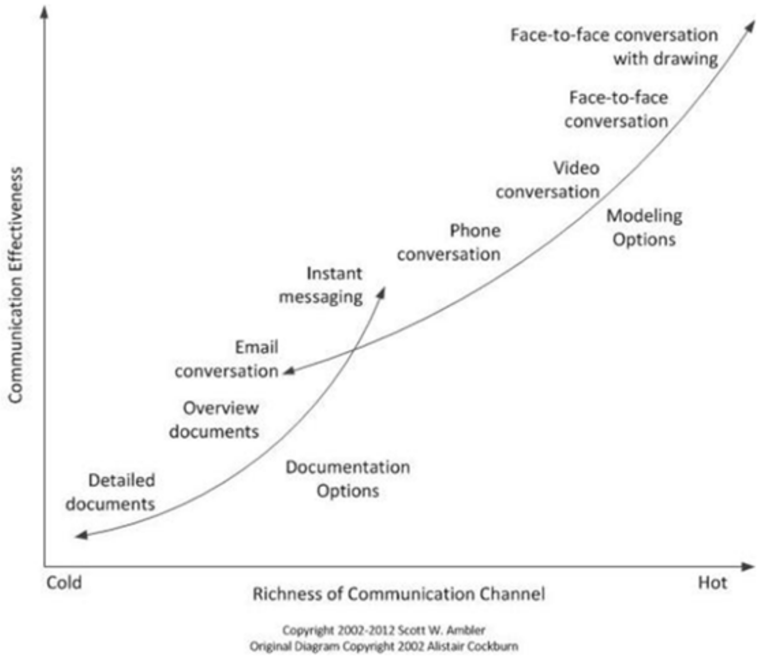


Figure 2.3. Richness of communication channel [Coc06].

Follow the processes. Adherence to processes is particularly important in distributed teams. Their compliance can reduce the costs associated with coordination and unnecessary communication, e.g., explaining the confusion. Profits from good communication are as follows:
— faster explanation of misunderstandings,
— information spreads throughout the team,
— quick response from the team to queries from the customers and the management,
— effective technical and business synchronisation,
— quick feedback in case of impediments and the ability to solve them.

Challenges in distributed environment Fortunately, the claim that running projects

in distributed teams is impossible is becoming less and less common. There are many examples not only that it is possible, but also that it works well [SS09, RCMX06]. However, the challenges are far from trivial. A distributed team can work but will have difficulty sustaining the same pace as a collocated team. According to Fowler [Fow06], distributed development requires more written documentation and, in general, more formal communication styles, than co-located teams do.

Technical challenges. There are plenty of opportunities to "improve" communication during remote meetings. Customers often choose low-budget solutions, such as Skype. The motivation, of course, is the desire to save money; however, it is often forgotten that wise investment in infrastructure not only increases comfort but also minimizes the risks in communication, which is reflected in the final success of the project. Therefore, it is recommended to use professional videoconferencing tools that allow for effective cooperation through desktop sharing, shared virtual table, etc...

Organizational challenges. Nowadays, there are many interesting solutions, ranging from simple software to videoconferencing hardware with excellent HD quality and the possibility of smooth transfer. The difficulty does not appear to be hardware limitations (costs), but organizational barriers. Customers of the so-called ”sensitive” sectors, such as the public sector, financial or even automotive sector no longer permit the use of video equipment in their factories or offices. The reasons are legal restrictions due to data protection. Another example of such a client is a telecommunications company, which provides processed and aggregated data about their customers for development purposes.

The human factor. Visualizing means realizing each other’s point of view. The ultimate goal is to increase mutual trust, which of course translates into the quality of the Project, including the efficiency and creativity of the team. You can visualize/”realize” the following aspects:
— work progress (where we are),
— the difficulties we struggle with,
— general morale of the team.

Awareness of each other and ways of visualizing our state In distributed teams,

we often have to deal with a mutual lack of trust, which increases over time. Each of the distributed teams may have incomplete knowledge about the progress in other teams. Let us remember that we are working on a common project and any conflict of interests is unacceptable, as well as different goals or focusing solely on the individual purposes of one team or part of a team. The typical solution in this situation is to create coordination functions; but firstly, it is not always enough, and secondly it generates additional costs. Therefore, we use tools that support mutual ”awareness” about the progress of work and shared vision. We achieve it in following ways:

— Backlog, which is an ordered list of functionalities. Of course, Backlog is shared not only for readout, but it is above all a platform for distributed management of requirements. This is a continuing activity, in which the Development Team and Product Owner work together on the details of user stories [SS13, pp. 12-13].
— Board (Scrum/Kanban), involves a fast visualization of the current work progress. The shared board shows the scope of work [KS10], which is interesting at the moment. Development Team (s) can better synchronize their work every day, guided by what the Board shows. It is also part of the system to respond to change, because this Board shows the limitations of the team.
— Wiki is something more than a static page with information or a project encyclopedia. Design Wiki is a platform for shared knowledge management as well as communication.
Each member of the Development Team is invited to participate in its creation, no matter where they are and regardless of their time zone!

We have good experience with the use of products by Atlassian: JIRA [Atl14a] - an issue tracking system and Confluence [Atl14b], one of the most popular wikis in corporate environments [SD12].

Pull and push communication. The communication media definition identifies the possible types of media that will aid the organization in delivering the messages in the communication program. It is best to use a combination of ”push” and ”pull” communication vehicles.

— Pull communication - places where information can be retrieved from at your audience’s leisure. We take only this information that interests us.
— Push communication - form of broadcasting, where you provide information directly to your audience. We inform and expect possible answers.

New ways of effective team communication. A new method of communication can be the so-called streaming information, such as Facebook, Google Plus or other Social Communication tools, such as Twitter or Yammer used at Capgemini [Cap12]. Obviously what we mean is using them in the context of software development project. Thanks to them we have a general view of what is happening with the project. Well-filtered information sometimes brings a lot more benefits than the most complete documentation.

There is nothing to hide here ... For the brave teams that want to (and can) be completely transparent, we offer mutual ”peeking”. This can be achieved in a simple way, by installing cameras and monitors in every design room. As a result, we have a general overview of the situation and a great support for other communication channels, such as telephone or email.

2.2.3. The architecture, development rules and domain knowledge

To avoid problems with the performance and motivation of teams, regardless of the previously proposed models, we pay attention to the following aspects of work in the project.

— *Shared ownership from the start.* An often erroneous assumptions is to build a strong team on the client side, and then creating a remote location. The best solution is the parallel development of teams or creating a team in a remote location on the basis of experienced team members, who previously worked onsite. They would provide knowledge to new people [CH04, p. 88 - 91].

— *Decide architecture together.* A vision of architecture is very important, and even in XP methodology its need is acknowledged [Bec00, p. 113]. The process of creating the system architecture should involve team members from all locations.

— *Get to know the client and domain.* A common problem due to the location is the lack of understanding of the domain and customer needs by team members. Domain knowledge is one of the most distinguishing factors supporting sound design decisions [CB10, p. 85]. It is good to ensure that the team has domain expertise and access to domain experts. Not enough context information off-site will lead to communication and development issues.

— *Form personal relationship.* Team members should know one another personally, especially since it is easier to overcome barriers formed in the case of remote communication.

— *Avoid local team taking aggressive ownership.* The onsite team, which has direct access to knowledge and the client would often take quick decisions on the basis of additional meetings organized adhoc. To avoid isolating the team working remotely, it is recommended to engage selected developers to regular meetings via video-conference.

A separate aspect involves the issues related to system design. We observe that software architecture distributes easily enough but the enterprise architecture often does not. If the chief architect works in a remote team, regular visits to the customer's premises are necessary for the current arrangements for enterprise architecture, especially in cases where other systems are not developed under agile methodology.

2.3. Summary

With all the advantages of the Agile! Nearshoring approach, such as cost advantage, scalability and high quality, the main challenge is the geographical and mental distance between the team members. This obstacle cannot be avoided, but can be minimized by applying proper tools and exchange programs, where people from each location travel and visit each other on a regular basis.
We decided to use several collaboration tools and to digitalize Scrum artifacts. The application of the proper tools can facilitate the Scrum processes and communication between the team members in the distributed work environments. In order to assure that every team member is aware of the status of the tasks and the status of the whole Sprint, we use social networking tools like blogs and Wikis in addition to videoconferencing and telephone.

2.3.1. About Capgemin and Nearshore Center Wrocław

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion. Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience and draws on Rightshore®, its worldwide delivery model.

Nearshore Center of Capgemini exists in Wroclaw since 2004, and is thus part of the worldwide Righshore® of delivery centers. More than 550 IT experts currently work in Wrocław delivering high quality services in the areas of software development, software package implementation and application life cycle services to German-speaking clients.

Capgemini in Poland employs more than 5000 consultants and IT as well as business process experts. Centers for IT and business process outsourcing services exist in Wroclaw, Kraków, Katowice and Opole with the main office serving the Polish market based in Warszawa.

References

[AL12] S.W. Ambler and M. Lines. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press. IBM Press, 2012.

[Atl14a] Atlassian. JIRA Documentation, 2014.

[Atl14b] Atlassian. Specification - Confluence Advanced Editor, 2014.

[Bec99] Kent Beck. Embracing Change with eXtreme Programming. *Computer*, 32(10):70–77, 1999.

[Bec00] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[Bec02] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[Bor08] J. Borg. *Body Language: 7 Easy Lessons to Master the Silent Language*. Pearson Prentice Hall Life, 2008.

[Cap12] Capgemini. Building the New Connected Enterprise - Capgemini case study, 2012.

[CB10] James O. Coplien and Gertrud Bjrnvig. *Lean Architecture: For Agile Software Development*. Wiley Publishing, 2010.

[CH01] Alistair Cockburn and Jim Highsmith. Agile Software Development: The People Factor. *Computer*, 34(11):131–133, 2001.

[CH04] James O. Coplien and Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[CHA11] CHAOS Manifesto. Technical report, The Standish Group International, 2011.

[Coc06] Alistair Cockburn. *Agile Software Development: The Cooperative Game (2nd Edition)*. Addison-Wesley Boston, 2006.

[Coh09] Mike Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 1st edition, 2009.

[Eva04] Eric Evans. *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.

[Fow06] Martin Fowler. Using an Agile Software Process with Offshore Development., 2006.

[Hal90] E.T. Hall. *The Silent Language*. Doubleday, 1990.

[KS10] H. Kniberg and M. Skarin. *Kanban and Scrum: Making the Most of Both*. Enterprise software development series. Lulu Enterprises Incorporated, 2010.

[Mar08] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1 edition, 2008.

[RCMX06] Balasubramaniam Ramesh, Lan Cao, Kannan Mohan, and Peng Xu.

Can Distributed Software Development Be Agile? *Commun. ACM*, 49(10):41–46, October 2006.

[SD12] Shiv Singh and Stephanie Diamond. *Social Media Marketing for Dummies*. John Wiley & Sons, 2012.

[SS09] Guido Schoonheim and Jeff Sutherland. Agile Distributed Development Done Right Using Fully Distributed Scrum, 2009.

[SS13] Ken Schwaber and Jeff Sutherland. The Scrum Guide. *Scrum. org, July*, 2013.

[WSG10] Elizabeth Woodward, Steffan Surdek, and Matthew Ganis. *A Practical Guide to Distributed Scrum*. IBM Press, 1st edition, 2010.

Chapter 3

Management of IT project experiences on the basis of SOEKS

IT project management is a major challenge for IT organizations which produce software for the needs of clients. The percentage of failed projects is still high - due to bad decisions, they are completed later than was planned in the schedule, they cost more than was assumed in the budget or do not fulfill the tasks specified in the contract. The experience of the authors clearly indicates the need to support project managers in their decision-making. Previous studies have aimed at building an intelligent system based on knowledge. It transpires, however, that due to the different specifications of projects and their strongly individual character, it is impossible to construct a single, objective knowledge base relying on rules and facts. Therefore, the authors have directed their attention towards an innovative method developed in recent years which involves recording experiences in the form of so-called SOEKS, and which may be the answer to the aforementioned problems. This chapter summarizes the first stage of this research.

3.1 Project management

Appropriate project management is one of the biggest challenges for today's IT organizations, each of which runs several or even a dozen different projects every year. These projects involve not only software development (though most commonly they do), but also a range of other activities such as building infrastructure, upgrading software or the implementation of new technologies in the organization. However, in every project, managers must make a number of specific decisions to ensure the success of the project entrusted to them (success being defined as the implementation of the project according to the agreed cost, schedule and scope). Due to the fact that every IT project is in definition a unique action (process) [PM08] it would seem that the decisions made by managers are different every time. However, the authors' observations in dozens of projects made it possible to isolate certain common elements (areas) in terms of which managers make decisions. This made it possible to find a common denominator between all projects and to isolate the "IT project environment", in which the authors include: the project team, the client, the tools and the methods applied during the project, as well as the expertise which supports the implementation processes of projects. The complexity of the "IT project environment" results that the process of decision-making must take into account a number of relationships between the identified elements (areas) of the environment [CK13].

How big is this problem? In the United States, more than $250 billion is being spent each year on IT application development of approximately 175 000 projects. The average cost of a development project for a large company is $2 322 000; for a medium company, it is $1 331 000; and for a small company, it is $434 000. A great many of these projects fails because they are in chaos. The Standish Group research shows a staggering 31.1% of projects will be canceled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars. One just has to look to the City of Denver to realize the extent of this problem. The failure to produce reliable software to handle baggage at the new Denver airport is costing the city $1.1 million per day [CH13].

This proves that bad decisions in IT project environment can cost a lot. Therefore, the another question arises whether in an such environment of a similar structure the same decisions should be made (and if it is at all possible). Does the new knowledge (organizational, technical) conceived as a result of completing a project (regarding, for example, taking specific steps to solve a given problem of an organizational nature (For example - the issue of gathering requirements in a conversation with a client) can be used in future projects? Can decisions made in one project be applicable (important) in other initiated projects? In other words - is it possible in the case of IT projects (unique by definition) to save knowledge and experience for the needs of future projects? This chapter and the research and experiments described in it are an attempt to answer this question.

3.1.1 Decision variables in IT projects

Consulting firms exploring factors influencing the ultimate success of IT projects point to the commitment of project management, the involvement of the client and the clarity of the defined requirements as most important [CH12]. However, these factors do not have a numerical measure and their importance is determined by a quantitative survey of the project participants. Research conducted by a team including the authors of this chapter has allowed the isolation of three aggregated variables (adequate to the success factors mentioned above) which affect the ultimate success of a project. Measurable values can be assigned to variables isolated by the authors, to allow project managers to take action (decisions) adequate to the values of these variables. These variables aggregate the key factors influencing the course of the project and these variables include:

- the maturity of the project team (how big is the level of its internal management and/or development processes' maturity; result of previous experiences of the team),
- the maturity of the client participating in the project (did the customer take part in similar projects, do the customer have full knowledge of their needs, can they specify all requirements etc.),
- the level of project entropy corresponding to the level of risk and the degree of clarification of the project scope (requirements).

These variables (as well as the possible ways of measuring them) have been discussed in detail in the book by Prof. Orlowski [OK12] and in previous articles by the authors [ZS13]; however, for the purposes of this chapter, these variables have been used as input parameters for one specific type of decision concerning the selection of the project management method. The decision regarding the selection of the project management method is one of the fundamental decisions made by managers. The choice of method determines the future course of the project. In general, these methods are divided into classic and agile. The classic ones are characterized by a significant role in planning processes and by central decision-making (done by the manager), while the agile ones are based on self-organizing teams focused on interaction, and dispersed (within the team), making decisions related to carrying out individual tasks. Hence, the decision of a project manager in terms of the management method of an IT project implies subsequent decisions regarding the division of labour, the way of reporting and measuring the effects, etc.

This chapter aims at presenting the possibility of recording project experiences based exactly on such decisions. In addition, for the purposes of this chapter, a certain simplification was adopted, assuming that the project decision regarding the method of project management and implementation is clear (i.e., the manager chooses a particular method). In reality, managers often use best project management practices derived from more than one method, adapting them to their projects. Best practices most commonly used by project managers primarily come from one of the aforementioned groups, i.e. classic or agile methods. This simplification results from the fact that this is the first stage of research on recording project experiences in the form of SOEKS. Ultimately, the intention of the authors is to record all kinds of project experiences (not just those relating to method selection) and to study their impact on the course of the project in order to then provide the possibility of using these experiences.

3.1.2 Relationships between project decisions and experiences

Project decisions may concern particular elements of the project environment (such as the assignment of tasks to people) and may relate to a higher level on which to manage the entire project (as in the case of the previously mentioned problem of selecting a method for the project).

Due to the fact that projects are inherently unique, it should be considered whether decisions made in one project can be significant for future projects. It would seem that due to the uniqueness of projects such action is not possible. If, however, it is taken into account that it is possible to isolate elements of the project environment - common for many projects (team, client requirements, information clarity level) - then the record of experiences, which shows the impact of these elements on the ultimate success of the project, can be a basis for making new decisions in new projects.

Given that the measure of the success of an IT project is still understood as its realization within the classical boundary conditions such as schedule, budget and scope, one can talk about the final evaluation of the impact of decisions made on this success. Having distinct variables (client maturity, team, entropy) and knowing what decision has been made under these conditions regarding the selection of project management methods, it is possible to talk about some experience to be used in future projects. However, this requires verifying whether, with the given values of the variables related to maturity and with that particular decision on the choice of the method, the project was a success. Then such project decisions can be recognized as experiences on the basis of which other project managers will be able to take further decisions on new projects. If a project manager is able to assess whether, with a specified level of project team maturity and client maturity, they applied a particular method of project management and the project was a success, then this decision will be regarded as a valuable experience for managers whose teams and clients exhibit similar levels of maturity.

It should be noted that one of the main characteristics of a good manager is the manager's intuition [T04] allowing for the right decisions to be made, which will probably never be replaced by an intelligent inference system. However, the recording and evaluation of decisions and experience resulting from project management can assist managers in making subsequent decisions in other projects. There are many concepts of knowledge-based systems, but they seem to be insufficient in the face of the project decisions issue, where experience also (or perhaps - most of all) plays a big role.

In this context the authors became interested in one of the newly developed methods, in management, of recording experiences (of any kind) - SOEKS. The purpose of this chapter is the adaptation of the assumptions of the SOEKS concept to the issues related to the management of IT projects. The next chapter presents the main assumptions of the concept.

3.2 SOEKS, a form of recording decision-making experiences

The experience of IT project managers in terms of their everyday decisions indicates that many of those decisions are uncertain in nature. Managers often rely on knowledge which is sometimes incomplete, or inaccurate. The consequence of such a state is obviously an increased risk of project failure. Undoubtedly, a support for decision-making with the help of intelligent solutions based on knowledge would be needed, which is the content of previous studies by the authors.

Classic solutions based on knowledge-based systems implement support for management decisions mostly relying on a knowledge base, which consists of facts and rules in the form of implications relating premises to conclusions. This follows directly from the assumptions of knowledge engineering, a branch of science which seeks to present intelligent human behavior through the integration of knowledge with computer inference systems in order to solve complex problems [SS07]. The concept of knowledge engineering has always involved attempts to imitate certain human behaviors such as learning, inference and prediction on the basis of available knowledge or obtained experience [CS13].

Classic knowledge-based systems (decision support systems, expert systems) are usually designed so that the user can achieve the result of a previously programmed inference based on gathered facts and rules. So far these tools have not provided the possibility of recording individual decisions which result from the application of this knowledge. The answer to this problem is to be the concept of Decisional DNA (DDNA), and in particular its main component, the so-called Set of Experiences, i.e. SOEKS.

SOEKS - *Set of Experience Knowledge Structure* can be described as a tool based on experience which is assumed to be capable of collecting and managing knowledge stored in a formal way. In particular, this refers to the knowledge of so-called decisional events, namely all the individual decisions which occur during the execution of a project and for which it is possible to specify both the premises and conclusions resulting from the adopted rules the decisions were based on.

The concept of SOEKS was thoroughly described in 2007 in the doctoral thesis of Cesar Sanin, PhD and it is part of the research on the so-called Knowledge Supply Chain System carried out at the University of Newcastle (Australia).

The assumptions that underpin this research say that managers facing daily decisional challenges rely largely on experience, and it is experience - not knowledge alone - that provides the basis for subsequent decisions. Therefore, if any decision was saved in a coherent and formal way, it could provide a processing basis for any inference engine. The SOEKS documentation is essentially the specifications of such a coherent structure.

Each SOEKS may consist of four components, as follows:

- variables (V) - are the representation of knowledge in the form of attribute-value pairs;
- rules (R) - are the record of inferences, they associate premises with the specific decisions which result from them;
- functions (F) - describe the relationship between variables;
- constraints (C) - are also a record of the relationship between variables in order to limit the possible solutions of decisional problems.

The SOEKS components and their mutual relationships are shown in Fig. 3.1.

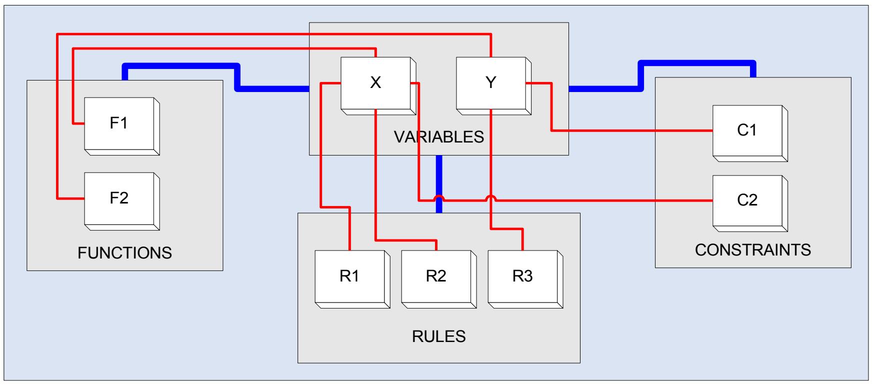
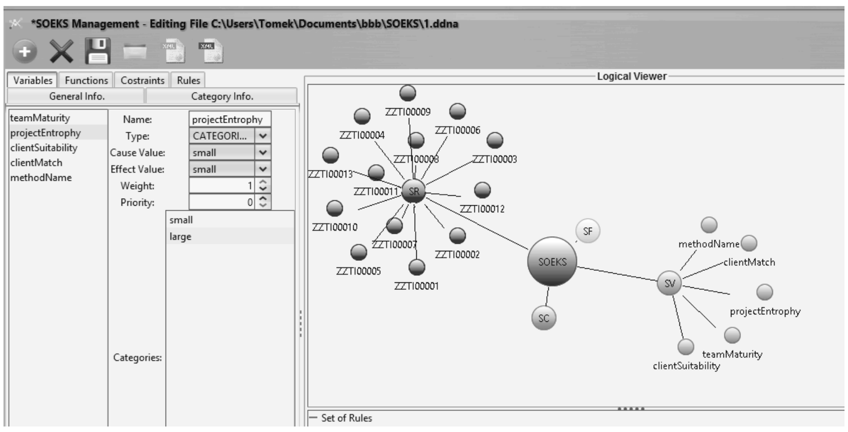


Figure 3.1. The SOEKS structure - four basic components and their mutual relationships

What confirms the validity of the approach represented by SOEKS are the various applications of this form of recording experiences in various fields of human activity, including medicine (the gathering and management of medical diagnoses to support everyday hospital practice) and transport (the analysis and selection of optimal rail routes). These implementations, however, show that

SOEKS works well mainly for domains in which the data regarding decisions is purely quantitative. The issue addressed by the authors of this study (the analysis of decisions in IT projects) unfortunately transgresses such a framework and also becomes a challenge for research on the applicability of SOEKS in this domain [SS09].

SOEKS is just a concept of recording knowledge, limited exclusively to the description of a structure. The authors do not promote any particular data format or methods of generating data. The only form of supporting the concept in terms of tools was initially a class set for Java language constituting API SOEKS. It is worth noting that for better management of gathered knowledge and experiences in SOEKS structures, it was decided that a dedicated tool - a decisional DNA manager, should be created [CT12].


*SOEKS Management - Editing File C:\Users\Tomek\Documents\bbb\SOEKS\1.ddna

Variables: teamMaturity, projectEntrophy, clientSuitability, clientMatch, methodName

Name: projectEntrophy
Type: CATEGORI...
Cause Value: small
Effect Value: small
Weight: 1
Priority: 0
Categories: small, large

Figure 3.2. DDNA Manager.

A DDNA Manager (*Decisional DNA Manager*) is a unique application which allows the easy construction of single decision-making structures, in particular entering, editing and storing the records of the decision-making experience, which can also be visualized (see Figure 3.2). The DDNA Manager was implemented in Java with the use of the aforementioned API SOEKS. As a result, in addition to manual operations, it also allows for integration with external solutions.

The authors decided to use the described possibilities offered by the concept of recording experiences to achieve the objective of their research, namely, the construction of a system to support project decisions made by IT project managers, through applying experiences from a variety of sources.

The following question can be raised: will the SOEKS idea be more effective than classic rule-based expert systems? This is very early stage of this project, therefore nothing can be proved for sure and is based mostly on assumptions and previous experiences. The authors have tried to accomplish the same goal – decision support for IT project managers – several times using various approaches. They were participating in a few projects run by international teams of researches and their results and gained experiences has always proved that something more than 'pure' knowledge must be stored and processes for reuse by project managers. Initial feasibility study gives hope (but it is not a proof yet) that SOEKS are the exact answers to this problem.

The following chapter elaborates on the assumptions adopted for such a project, clarifies the objectives and presents the course of the carried out experiment.

3.2.1 Using SOEKS to represent project decisions

Is the SOEKS concept adequate to represent project decisions? The past experience of the Team justifies the need to answer this question. The constantly growing number of IT projects and the considerable proportion of those which fail prove the aforementioned thesis about the growing complexity of the environment in which decision makers operate. This, in turn, was an indication for the authors to undertake research in this direction. The main goal was to build a knowledge-based intelligent solution for supporting decision-making in the IT organization. The observations and experiments conducted over the years show that the classic system based on knowledge (stored as a combination of rules and facts) is not sufficient [CO11, SO07]. Projects in this sector have a very specific and individual character. It is not possible to build a completely objective knowledge base for the needs of such a solution. Therefore, it seems necessary to collect knowledge from many different sources, in particular - from various Project Managers from various projects executed by them and concerning various aspects of decision-making. It is thus necessary to collect and process not so much knowledge but experiences, which must be continually assessed.

In spite of such diversity, it is necessary to distinguish certain common aspects - perhaps at the expense of certain simplifications. The analysis of some cases led the authors to the conclusions depicted in Figure 3.3. Managing IT projects involves the simultaneous management of several different aspects of those projects. These include: cooperation with experts, the maturity of the project team, the relationship with the client, the selection of tools and methods of project management, as well as applying the developed best practices. These aspects should be treated as decision variables.

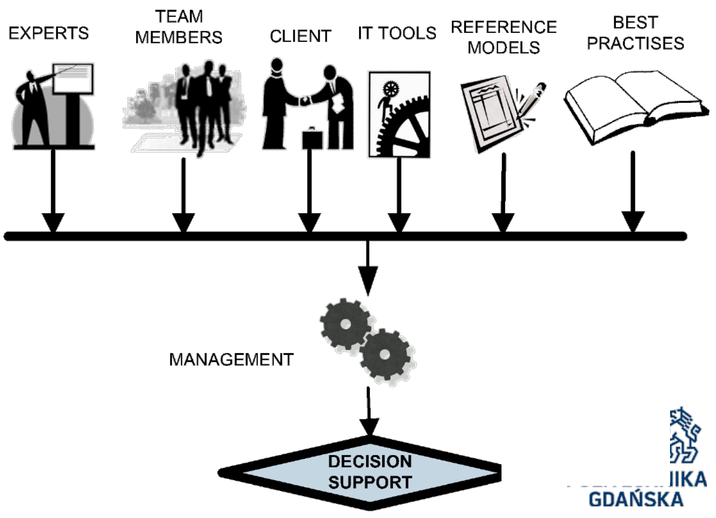


Figure 3.3. IT project structure.

A research project entitled "*Smart Modeling Approach of IT Project Management for Small and Medium Companies*" has become the framework for research based on the above premises. It is carried out by three partners: the University of Newcastle (Australia), the VicomTech Institute (Spain) and the Technical University of Gdansk (Poland), based on funding from European funds allocated to the FP7-PEOPLE-2009-IRSES program. The main objective of the project is the development of a support system for decision-making in IT projects with the use of SOEKS as a means of formalizing knowledge and experiences.

For the needs of the project, the following assumptions about its essence were made:

— In the basic version, the system will support the most crucial decision of the IT project manager - selecting the methodology for conducting the project (the output variable of the system - output). The set of available methodologies/frameworks will be closed at the initial stage. Possible values include: PRINCE2 (typical heavy method of management level), RUP (typical heavy method for development teams), XP (example of agile method), Scrum (agile method/framework).

— The choice of methodology will depend on four values (input variables - input): the maturity of the project team, (lack of) order (entropy) of the project, the client's maturity and their suitability to the specific nature of the project.

The authors are aware that such distinction of methods is rather simplified and may be treated as disputable. There are opinions that RUP is not heavy method (e.g. because not all of its artifacts are strongly required). Also distinction PRINCE2 vs. Scrum is not that clear. Both can be used concurrently and support each another; they are compatible and project manager may not to be forced to chose one of the other. Thus, according to the authors such decision should be interpreted as the answer to the question whether given project required high-level management (then PRINCE2 is the best choice) or the focus on development is the most critical (Scrum is recommended so).

From the point of view of the possibilities offered by SOEKS, it was assumed that:

— At the initial stage of the project, variables and rules will be used to record individual experiences/decisions; functionalities and limitations will not be taken into account.

— Apart from the above-mentioned input variables (representing the body of rules) and the output variable (conclusion-decision), additional variables were used such as the ID of the project (many decisions are allowed for one project), the ID of the expert entering data into the database (many decisions can be obtained from one expert), the comment of the expert and the evaluation of the decision taken (available after its implementation).

Table 3.1. Variables and rules applied in SOEKS.

Variables	Rules
Team Maturity (input)	IF Team Maturity = <value> AND Project Entropy = <value> AND Client Suitability = <value> AND Client Match = <value> THEN PM Method = <value>
Project Entropy (input)	
Client Suitability (input)	
Client Match (input)	
PM Method (output)	
Project ID (aux)	
Expert ID (aux)	
Comment (aux)	
Success Rate (aux)	

It was assumed that one SOEKS is a representation of one project decision, i.e. the experience acquired from one Project Manager regarding one given project. Each initial decision will therefore be written as a set of variables and a single rule. The only variable which initially (during the construction phase of SOEKS) will not be assigned a value is the *Success Rate*, since the evaluation of the decision will be made after a specified time. Before the project is initiated, the expert enters their first experience - a decision regarding the current methodology of the project. This is called an incomplete experience.

Examples of rules have been presented below.

Example rule #1:

```
If (
CLASS ITProject with the PROPERTY teamMaturity EQUALS TO initial
AND
CLASS ITProject with the PROPERTY projectEntrophy EQUALS TO large
AND
CLASS ITProject with the PROPERTY clientSuitability EQUALS TO no
AND
CLASS ITProject with the PROPERTY clientMatch EQUALS TO yes
)
then
( CLASS ITProject with the PROPERTY methodName EQUALS TO PRINCE2 )
```

Example rule #2:

```
If (
CLASS ITProject with the PROPERTY teamMaturity EQUALS TO initial
AND
CLASS ITProject with the PROPERTY projectEntrophy EQUALS TO large
AND
CLASS ITProject with the PROPERTY clientSuitability EQUALS TO no
AND
CLASS ITProject with the PROPERTY clientMatch EQUALS TO yes
)
then
( CLASS ITProject with the PROPERTY methodName EQUALS TO Scrum )
```

It is assumed that in the course of the project, or after its completion, the expert can perform the following activities:

- evaluate their initial decision (fill in the value of the Success Rate variable, due to which incomplete experiences will change their status to complete experiences),
- enter subsequent experiences regarding the carried out project.

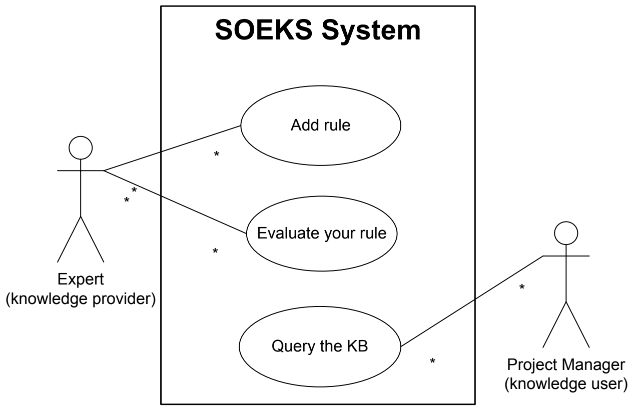


Figure 3.4. Three main use cases of the proposed solution.

In the first version of the proposed solution, apart from the expert, the authors see a second actor - the user (see Figure 3.4), who will be able to apply the basic functionality of the system, namely ask a question to the SOEKS database, describing the realities of their project (i.e., giving appropriate values to the input variables). The system will return the experiences of other experts included in the base, categorizing them in terms of matching.

3.3 Implementation

The concept of recording project experiences, presented above, regarding decisions on the selection of project management methods can be applied as long as there is access to a preview of experiences from earlier projects. Hence, the authors believe that the best way of applying the SOEKS idea is by implementing it in the tools supporting project management. This is due to the fact that it is common practice for project managers to use specific tools to assist in the management of a project. The tools used by managers allow such tasks as defining teams, defining workflows or creating project documentation.

3.3.1 Applying SOEKS in project management tools

Some of the popular project management tools (such as IBM Rational Team Concert) have process templates customized for projects carried out both according to classic approaches, as well as agile ones. In this way, project managers can trace the progress of projects in terms of the chosen method of management. Such an approach is consistent with the pre-set assumptions that managers depend on some initial parameters (maturity), to decide how to conduct the project (project management method). The inclusion of such a decision in a tool, saving it and then evaluating its effect can be a valuable experience benefiting future projects. Hence, the authors are exploring the possibility of implementing the SOEKS concept in an exemplary project management tool so that the experience of previous projects can be fully used in future projects.

With this approach, it is possible to gather decisions and their consequences (i.e. experiences) for subsequent processing for future projects. This approach also gives managers making decisions during subsequent projects a reference point by showing the consequences of certain decisions.

3.3.2 Suggested implementation – further steps

In the first step, the authors selected a tool with the ability to define projects according to classic and agile principles.

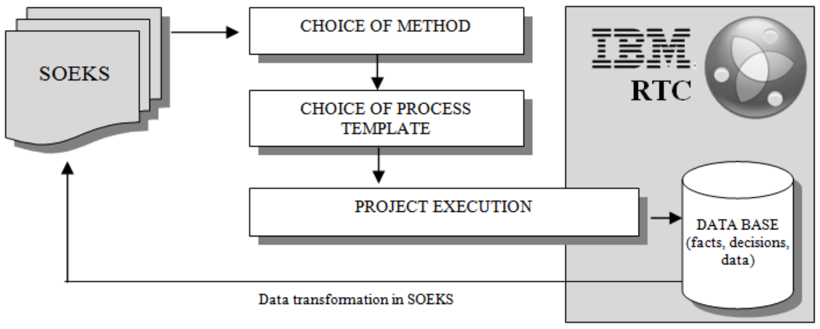

SOEKS
CHOICE OF METHOD
CHOICE OF PROCESS TEMPLATE
PROJECT EXECUTION
IBM RTC
DATA BASE (facts, decisions, data)
Data transformation in SOEKS

Figure 3.5. Diagram of relationships and the flow of information in the RTC-plugin SOEKS system.

In the next step, there are plans to verify to what extent the decisions recorded in the tool will be converted to a record compliant with SOEKS.

Then, the mechanism of processing experiences will be applied, regarding also the way the experiences are read by managers making decisions in subsequent projects.

In the final step, a plug-in for a selected tool will be built which will form a kind of expert system allowing for a method of project management to be suggested on the basis of experiences.

3.4 Conclusion

The approach presented in this chapter aims at streamlining decision-making processes in IT projects. It is extremely valuable in such processes to consider the experiences and knowledge created (acquired) during the execution of previous projects. Firstly, the authors verified how far the SOEKS concept, popular in many industries (medicine, transport), may be useful to record a strictly managerial experience. After concluding the analysis with positive findings (consulted with the authors of the SOEKS concept) it was established that the experiences should be recorded constantly in the course of the project. Hence, the authors undertook the project to show the possibility of a continuous record of decisions (project experiences) by implementing the SOEKS concept in an IT tool for supporting project management.

The following question can be raised – would such approach be suitable for each project manager and every IT project? Yes, that is the assumption. But it must be taken into consideration that this is additional job which costs time and energy. Gathering knowledge and experience in small project (e.g. a few people involved) may be rather easy for single PM. However, the process of transforming knowledge to SOEKS within big projects will require cooperation of various roles playing part – system architects, team leaders, domain experts and even programmers (regardless which methods will be used). That is why we can say that the whole concept of SOEKS may be support for the larger group of people involved in project.

What are the next steps? There are plans to develop mechanisms to process project experiences stored in the form of SOEKS in such a way that managers undertaking the realization of new projects would automatically be able (using a dedicated plug-in) to obtain guidelines stating which mistakes to avoid, and which methods (and best practices) of management should be selected for a given project.

The research discussed in the chapter is funded by the FP7 Marie Curie International Research Staff Exchange Scheme (FP7-PEOPLE-2009-IRSES); project name: Smart multipurpose knowledge administration environment for intelligent decision support systems development 'SASD'; subtask name: Smart Modeling Approach of IT Project Management for Small and Medium Companies (GUT, Poland).

References

[PM08] PMI. *A Guide to the Project Management Body of Knowledge*, *Fourth Edition*, 2008.

[CK13] Adam Czarnecki, Liliana Klich and Cezary Orłowski. *Simulation of the IT Service and Project Management Environment*, vol. LXII, issue 1/2013, 161-180, 2013.

[CH12] The CHAOS Manifesto, The Standish Group, 2012.

[OK12] Cezary Orłowski and Zdzisław Kowalczuk, *Modelowanie procesów zarządzania technologiami informatycznymi*, 2012.

[ZS13] Artur Ziółkowski and Tomasz Sitek. Projekt-czynnik-decyzja. Badanie czynników decyzyjnych w projektach informatycznych i ich wpływu na powodzenie projektów. *Zarządzanie projektami i modelowanie procesów*, 49-61, 2013.

[T04] Andrzej Tubielewicz. *Zarządzanie strategiczne w biznesie międzynarodowym*, 2004.

[SS07] Cesar Sanin, Edward Szczerbicki and Carlos Toro. An OWL Ontology of Set of Experience Knowledge Structure. *Journal of Universal Computer Science*, vol. 13, Issue 2, 209-223, 2007.

[CS13] Adam Czarnecki and Tomasz Sitek. Ontologies vs. Rules — Comparison of Methods of Knowledge Representation Based on the Example of IT Services Management. *Information Systems Architecture and Technology: Intelligent Information Systems, Knowledge Discovery, Big Data and High Performance Computing*, 99-109, 2013.

[SS09] Cesar Sanin and Edward Szczerbicki. Experience-based Knowledge Representation: SOEKS. *Cybernetics and Systems: An International Journal*, vol. 40, 99-122, 2009.

[CT12] Cezar Sanín, Carlos Toro, Haoxi Zhang, Eider Sanchez, Edward Szczerbicki, Eduardo Carrasco, Wang Peng and Leonardo Mancilla-Amaya. Decisional DNA: A multi-technology shareable knowledge structure for decisional experience. *Neurocomputing,* vol. 88, 42–53, 2012.

[CO11] Adam Czarnecki and Cezary Orłowski. Application of Ontology in the ITIL Domain. *Information Systems Architecture and Technology : Service Oriented Networked Systems*, 99-108, 2011.

[SO07] Tomasz Sitek and Cezary Orłowski. Evaluation of information technologies - concept of using intelligent systems. In: *Information systems architecture and technology: application of information technologies in management systems*, 217-224, 2007.

PART II
REQUIREMENTS ENGINEERING

Chapter 4

Functional safety, traceability, and Open Services

There are clear business reasons why the achievement of the functionality of more and more industrial products is definitely shifted to the use of embedded software replacing earlier hardware solutions. It is commonly known that the elicitation, identification, analysis, specification, modeling, validation, and management of the requirements in general and of functional safety requirements in particular is a demanding process with special implications for the case of software also reflected in related general and industry specific standards. This chapter gives a brief overview of standard approaches to functional safety with examples from the medical domain. It is highlighted that one of the key requirements of all approaches is traceability. The emerging Open Services for Lifecycle Collaboration (OSLC) technology, exploiting the architecture of the World Wide Web, is shown to have a determining impact on the future of the satisfaction of traceability requirements of safety-critical software development among others.

4.1 Functional Safety

It is unfortunate but also unquestionable that there is a risk that any product, with embedded software or not, causes harm to the health of people directly or indirectly as a result of damage to property or to the environment. The risk is usually defined as the product of the probability of occurrence and of the severity of the harm. There are of course products whose risk of causing harm is acceptable in a given context, based on the current values of society (IEC 61508). Safety-critical systems, on the other hand, have so high risk of causing harm that this risk must be reduced to a level "as low as reasonably practicable" (ALARP) required by ethics and regulatory regimes. Functional safety is the freedom from unacceptable risk regarding the functionality of the system.

The growing expectations regarding software components of safety-critical systems is a consequence of the changing impact of software on the consumer value of electrical, electronic or programmable electronic (E/E/PE) systems (IEC 61508) which was earlier fundamentally determined by hardware components with software primarily used for algorithmic tasks. Increasingly, embedded software is creating competitive differentiation for manufacturers [Ba11] in many industries including Automation, Aerospace, Automotive, Rail, Medical, Machinery, Nuclear, Process Automation and Consumer Products.

Software is perceived by business as more capable to be adapted to fluid requirements changes than hardware. In the software engineering discipline, it has for long been recognized however that the only way to achieve high reliability is to follow appropriately defined processes [BT95] [BT99] [BM00]. The necessary processes are summarized in international standards (ISO/IEC 12207 for software, ISO/IEC 15288 for systems), while their assessment and improvement is facilitated by the ISO/IEC 15504 series of standards (SPICE) currently evolving into the ISO/IEC 330xx series.

Regarding medical systems for example, the Association for the Advancement of Medical Instrumentation (AAMI) software committee reviewed ISO/IEC 12207:1995 and identified a number of shortcomings due to the fact that it was a generic standard. As a result a decision was taken to create a new standard which was domain specific to medical device software development, and in 2006, the new standard IEC 62304:2006 Medical device software - Software life cycle processes, was released. IEC 62304:2006 is approved today by the FDA (U.S. Food and Drug Administration) and is harmonized with the European MDD (Medical Device Directive). The quality management standard ISO 13485:2003, and the risk management standard ISO/IEC 14971:2007 are considered to be aligned with IEC 62304:2006 and their relationship is documented in IEC 62304:2006 itself. An extensive revision of the ISO/IEC 12207 standard took place in its release in 2008. As a result, all derived standards, including IEC 62304:2006, are under review.

To facilitate the assessment and improvement of software development processes for medical devices, the MediSPICE model based upon ISO/IEC 15504-5 was developed and has been renamed to MDevSPICE® in 2014 [LCMC14], [MCCL14]. The Process Reference Model (PRM) of MDevSPICE® will enable the processes in the new release of IEC 62304 to be comparable with those of ISO 12207:2008 [CMC13]. The above points give just a glimpse of the changes heavily affecting software developers in the medical devices domain.

Instead of containing actual recommendations of techniques, tools and methods for software development, IEC 62304 encourages the use of the more general IEC 61508-3:2010 Functional Safety of Electrical/Electronic/ Programmable Electronic Safety-related Systems – Part 3: Software requirements as a source for good software methods, techniques and tools.

4.2 Traceability

Traceability and even bilateral (ISO/IEC 15504) or bidirectional (CMMI) traceability are key notions of all process assessment and improvement models. [MCCS12] reports about an extensive literature review which classifies the models involving software traceability requirements according to the scope of the model, that is:

— Generic software development and traceability including CMMI and ISO/IEC 15504.

- Safety-critical software development and traceability including DO-178B (Software Considerations in Airborne Systems and Equipment Certification) and Automotive SPICE.
- Domain specific software traceability requirements which, in the case of medical devices for example, include IEC 62304 (Medical Device Software – Software Life Cycle Processes), MDD 93/42/EEC (European Council. Council directive concerning medical devices), Amendment (2007/47/EC), US FDA Center for Devices and Radiological Health Guidances, ISO 14971:2007. (Medical Devices – Application of Risk Management to Medical Devices), IEC/TR 80002–1:2009 (Medical Device Software Part 1: Guidance on the Application of ISO 14971 to Medical Device Software), and ISO 13485:2003 (Medical Devices – Quality Management Systems – Requirements for Regulatory Purposes)

Let us first consider the definitions of traceability in the following broadly known models.

- CMMI® for Development, Version 1.3 November 2010
  - Traceability: A discernable association among two or more logical entities such as requirements, system elements, verifications, or tasks.
  - Bidirectional traceability: An association among two or more logical entities that is discernable in either direction (i.e., to and from an entity).
  - Requirements traceability: A discernable association between requirements and related requirements, implementations, and verifications.
- Automotive SPICE® Process Assessment Model, v2.5 2010-05-10 using the definition originally adopted in IEEE Std 610.12-1990 Standard Glossary of Software Engineering Terminology
  - Traceability: 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.

Automotive SPICE® systematically requires bilateral traceability in all of the following engineering base practices:

- ENG.1 Requirements elicitation
- ENG.2 System requirements analysis
- ENG.3 System architectural design

- ENG.4 Software requirements analysis
- ENG.5 Software design
- ENG.6 Software construction
- ENG.7 Software integration test
- ENG.8 Software testing
- ENG.9 System integration test
- ENG.10 System testing

It is important to highlight that traceability has been considered as a key issue by the agile community as well. Scott Ambler, one of the key personalities of the agile movement, states in 1999 that “My experience shows that a mature approach to requirements traceability is often a key distinguisher between organizations that are successful at developing software and those that aren’t. Choosing to succeed is often the most difficult choice you’ll ever make—choosing to trace requirements on your next software project is part of choosing to succeed.” [A99]

Scott Ambler’s advice in 2014 [A14]:
“Think very carefully before investing in a requirements traceability matrix, or in full lifecycle traceability in general, where the traceability information is manually maintained.”
He also describes rational arguments which support that maintaining traceability information makes sense in the following situations:

- Automated tooling support exists
- Complex domains
- Large teams or geographically distributed teams
- Regulatory compliance

The already cited IEC 61508 standard lists forward traceability as well as backward traceability as recommended and even highly recommended at the safety integrity levels 3 and 4 (SIL 3 and 4). The safety integrity of a system can be defined as "the probability (likelihood) of a safety-related system performing the required safety function under all the stated conditions within a stated period of time„. Safety integrity levels are defined by probability (required likelihood) of failure which is the inverse of safety integrity.

In IEC 61508, “highly recommended” means that if the technique or measure is not used then it is the rationale behind not using it which has to be carefully demonstrated during safety planning and assessment.

Unfortunately, traceability as well as the actual assessment of the satisfaction of the crucial traceability requirements is difficult to achieve with the heterogeneous variety of application lifecycle management (ALM) tools companies are faced with [PRZ09], [MD12]. Using a manual approach, assessors can only recur to sampling which has ultimate weaknesses:

- Traceability is basically restricted to the closed ALM system. Representational State Transfer (REST) APIs are mostly available for providing internal data. However, a standardized open form of exchange is only made possible by the below described OSLC approach.
- Useful traceability reports can be generated, but they are static while requirements and identified defects are very dynamically changing artifacts, and may even originate from outside the ALM system.
- Assessors and users may be easily confused by the complexity of the set of widgets, such as buttons, text fields, tabs, and links which are provided to access and edit all properties of resources at any time.
- Assessors and users need to reach destinations such as web pages and views by clicking many links and tabs whose understanding is not essential for the assessment.

4.3 Open Services

Considering all of the above discussion, the need for the automation of assessing and maintaining traceability is imminent. It is this automation to which the Open Services for Lifecycle Collaboration (OSLC) [OSLC08] initiative opens the way.

OSLC is the recently formed cross-industry initiative aiming to define standards for compatibility of software lifecycle tools. Its aim is to make it easy and practical to integrate software used for development, deployment, and monitoring applications. This aim seems to be too obvious and overly ambitious at the same time. However, despite its relatively short history starting in 2008, OSLC is the only potential approach to achieve these aims at a universal level, and is already widely supported by industry. The OSLC aim is of course of utmost significance in the case of the safety-critical systems. The unprecedented potential of the OSLC approach is based on its foundation on the architecture of the World Wide Web unquestionably proven to be powerful and scalable and on the generally accepted software engineering principle to always focus first on the simplest possible things that will work.

The elementary concepts and rules are defined in the OSLC Core Specification [OCS13] which sets out the common features that every OSLC Service is expected to support using the terminology and generally accepted approaches of the World Wide Web Consortium (W3C). One of the key approaches is Linked Data being the primary technology leading to the Semantic Web which is defined by W3C as providing a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. And formulated at the most abstract level, this is the exact goal OSLC intends to achieve in the interest of full traceability and interoperability in the software lifecycle. OSLC is having a determining cross-fertilizing effect on the progress of the more general purpose Semantic Web itself.

The OSLC Core Specification is actually the core on which all lifecycle element (domain) specifications must be built upon. Examples of already defined OSLC Specifications include:

- Architecture Management
- Asset Management
- Automation
- Change Management
- Quality Management
- Requirements Management

Regarding for example the Requirements Management Specification whose version 2.0 was finalized in September 2012 [ORM12], it builds of course on the Core, briefly introduced above, to define the resource types, properties and operations to be supported by any OSLC Requirements Definition and Management (OSLC-RM) provider. Examples of possible OSLC Resources include requirements, change requests, defects, test cases and any application lifecycle management or product lifecycle management artifacts. Resource types are defined by the properties that are allowed and required in the resource. In addition to the Core resource types (e.g. Service, Query Capability, Dialog, etc...), the minimal set of special Requirements Management resource types simply consists of:

- Requirements
- Requirements Collections.

The properties defined in the OSLC Requirements Management Specification describe these resource types and the relationships between them and all other resources. The relationship properties describe for example that

- the requirement is elaborated by a use case
- the requirement is specified by a model element
- the requirement is affected by another resource, such as a defect or issue
- another resource, such as a change request tracks the requirement

- another resource, such as a change request implements the requirement
- another resource, such as a test case validates the requirement
- the requirement is decomposed into a collection of requirements
- the requirement is constrained by another requirement

Regarding the Change Management Specification, its version 3.0 is under development in 2014, and builds of course on the Core, briefly mentioned above, to define the resource types, properties and operations to be supported by any OSLC Change Management (OSLC CM) provider.

Examples of possible OSLC CM Resources include defect, enhancement, task, bug, activity, and any application lifecycle management or product lifecycle management artifacts. Resource types are defined by the properties that are allowed and required in the resource.

The properties defined in the OSLC Change Management Specification describe these resource types and the relationships between them and all other resources. The relationship properties describe in most general terms for example that

- the change request affects a plan item
- the change request is affected by a reported defect
- the change request tracks the associated Requirement
- the change request implements associated Requirement
- the change request affects a Requirement

OSLC is currently at the technology trigger stage along its hype cycle [FR08] [B09], it is already clear however that it is the approach which has the potential to have a determining impact on the future of the satisfaction of the traceability requirements of safety-critical software development among others.

Full traceability of a requirement throughout the development chain and even the entire supply chain is also a major focus point of the recently completed authoritative European CESAR project (Cost-Efficient Methods and Processes for Safety Relevant Embedded Systems) which adopted interoperability technologies proposed by the OSLC initiative.

Another important European project, completed in 2013 and exploiting OSLC, is iFEST (industrial Framework for Embedded Systems Tools).

4.4 Conclusion

The chapter has shown that all approaches to achieving functional safety require the establishment of bilateral traceability between development artifacts. Unfortunately, the manual or even tool supported creation and maintenance of traceability is currently difficult and expensive. The emerging industry supported OSLC initiative has been shown to have a determining impact on the future of the satisfaction of traceability requirements of safety-critical software development among others.

References

[A14] S. Ambler, Agile Requirements Best Practices, 2014. http://www.agilemodeling.com/essays/agileRequirementsBestPractices.htm (accessed: 15/08/2014).

[A99] S. Ambler, Tracing Your Design. Dr.Dobb's Journal: The World of Software Development, 1999.

[B09] M. Biro. The Software Process Improvement Hype Cycle. Invited contribution to the Monograph: Experiences and Advances in Software Quality (Guest editors: D.Dalcher, L. Fernndez-Sanz) CEPIS UPGRADE Vol. X (5) pp. 14-20 (2009) http://www.cepis.org/files/cepisupgrade/issue%20V-2009-fullissue.pdf (accessed: 15/08/2014)

[B14] M. Biro. Open services for software process compliance engineering. In V. Geffert, B. Preneel, B. Rovan, J. Štuller, & A. Tjoa (Eds.) SOFSEM 2014: Theory and Practice of Computer Science, vol. 8327 of Lecture Notes in Computer Science, (pp. 1–6). Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-04298-5_1 (accessed: 15/08/2014)

[Ba11] M. Bakal. Challenges and opportunities for the medical device industry. IBM Software, Life Sciences, Thought Leadership White Paper, (2011)

[BM00] M. Biró, R. Messnarz R. Key success factors for business based improvement. Sofware Quality Professional 2:(2) pp. 20-31, 2000. http://asq.org/pub/sqp/past/vol2_issue2/biro.html (accessed: 15/08/2014)

[BR98] M. Biró, T. Remzső. Business motivations for software process improvement. ERCIM NEWS 32: pp. 40-41, 1998. http://www.ercim.eu/publication/Ercim_News/enw32/biro.html (accessed: 15/08/2014)

[BT95] M. Biró, P. Turchányi. Software Process Assessment and Improvement from a Decision Making Perspective. ERCIM NEWS 23: pp. 11-12, 1995. http://www.ercim.eu/publication/Ercim_News/enw23/sq-sztaki.html (accessed: 15/08/2014)

[BT99] M. Biró, C. Tully. The software process in the context of business goals and performance. In: Messnarz R, Tully C (ed.) Better Software Practice for Business Benefit: Principles and Experience. 409 p. Washington; Paris; Tokyo: Wiley - IEEE Press, 1999. pp. 15-28. (ISBN: 978-0-7695-0049-2)

[CMC13] V. Casey, F. McCaffery. The development and current status of MediSPICE. In T. Woronowicz, T. Rout, R. OConnor, A. Dorling (Eds.) Software Process Improvement and Capability Determination, vol. 349 of Communications in Computer and Information Science, (pp. 4960). Springer Berlin Heidelberg, 2013.

[FR08] J. Fenn, M. Raskino. Mastering the Hype Cycle. Harvard Business Press, 2008.

[LCMC14] M. Lepmets, P. Clarke, F. McCaffery, A. Finnegan, A. Dorling. Development of a Process Assessment Model for Medical Device Software Development. In: industrial proceedings of the 21st European Conference on Systems, Software and Services Process Improvement (EuroSPI 2014), 25-27 June, Luxembourg. (2014)

[MCCL14] F. McCaffery, P. Clarke, M. Lepmets. Bringing Medical Device Software Development Standards into a single model - MDevSPICE. To appear in: Irish Medicines Board Medical Devices Newsletter Vol 1(40). (2014)

[MCCS12] F. McCaffery, V. Casey, M.S. Sivakumar, G. Coleman, P. Donnelly, J. Burton. Medical device software traceability. In J. Cleland-Huang, O. Gotel, & A. Zisman (Eds.) Software and Systems Traceability, (pp. 321-339). Springer London, 2012. http://dx.doi.org/10.1007/978-1-4471-2239-5_15 (accessed: 15/08/2014)

[MD12] T.E. Murphy, J. Duggan. Magic Quadrant for Application Life Cycle Management. Gartner, 2012.

[OCS13] Open Services for Lifecycle Collaboration Core Specification Version 2.0. (2013) http://open-services.net/bin/view/Main/OslcCoreSpecification (accessed: 15/08/2014)

[ORM12] Open Services for Lifecycle Collaboration Requirements Management Specification Version 2.0, 2012. http://open-services.net/bin/view/Main/RmSpecificationV2 (accessed: 15/08/2014)

[OSLC08] Open Services for Lifecycle Collaboration,2008. http://open-services.net/ (accessed: 15/08/2014)

[PRZ09] G. Pirklbauer, R. Ramler, R. Zeilinger. An integration-oriented model for application lifecycle management. Proceedings of the 11th International Conference con Enterprise Information Systems (ICEIS 2009), pages 399-403, INSTICC. (2009)

[RBMC14] G. Regan, M. Biro, F. Mc Caffery, K. Mc Daid, D. Flood. A traceability process assessment model for the medical device domain. In B. Barafort, R. O'Connor, A. Poth, & R. Messnarz (Eds.) Systems, Software and Services Process Improvement, vol. 425 of Communications in Computer and Information Science, (pp. 206–216). Springer Berlin Heidelberg, 2014. http://dx.doi.org/10.1007/978-3-662-43896-1_18 (accessed: 15/08/2014)

Chapter 5

The sufficient criteria for consistent modeling from the context diagram to the business use case diagrams driven by consistency rules

It would be unreasonable to capture everything we want in a single comprehensive diagram. Such diagram would be too big and complex, and primarily too confusing to display all architectural concerns. Key software architecture aspects should be therefore captured in several diagrams or even models, while the architectural details are grouped in adequate views and presented in a set of logically linked subsequent models. Preferably all these subsequent models should be logically originated from a certain initiating diagram. We propose to start modeling the software architecture from the initial business context diagram with subsequent process decomposition diagrams, and relevant business use case diagrams. The verification of consistency of all these diagrams would be obviously needed to identify all omissions and errors in requirements most preferably at the early stage of the development process. Semi-formal nature of these diagrams makes this consistency verification challenging. We propose to apply simple consistency rules to enable automatic verification of consistency of diagrams presenting business context, and process decomposition, and business use case models. Moreover these consistency rules enable simultaneous modeling of the functionality, of the structure and of the behavior of the business layer of the software architecture.

Typical communication problems arise when some specific software should be developed within not negotiable and strict timeframe and cost constraints. It usually occurs in public sector development projects when business needs are not sufficiently defined and technical documentation is missing or at least obsolete while majority of technology platforms was already specified and finally fixed in Terms of References. In such circumstances effective capture of the most important business aspects and establishing an efficient communication highway with business owners are crucial for the overall success of the software development process. [1]

To complement such picture of the whole analysis and development scenario the contracting institution usually is managing no Enterprise Architecture or

[1] Interestingly to note that above problems seem perpetual despite the rapid development of ITC technologies, their availability and vast potential. The problems are of very similar nature despite they are arising for a case of quite new function capable system or now even more often for a case of platform modernisation project. In the latter case usually the existing platform of spaghetti like architecture supposed to support several business functionalities should be analysed and significantly improved with removal of identified bottle-necks and with optimised maintenance features while adding newly emerging functionalities.

TOGAF [TOGAF] artifacts exist, while all important details of AS-IS ICT (Information and Communications Technology) infrastructure are residing in IT staff's brains rather than in process maps, data models and infrastructure schemes. Sometimes even most of IT staff brains belong to the outsourcing company competing and as such lost or excluded from the finalized tender for the development/modernization concerned.

In object-oriented software development, the UML [UML] has become the standard notation for the software architecture modeling at different stages of the life cycle and at different views of the software system, including the specification of requirements. Thus in the majority of projects using UML diagrams [CY02], [K08], use case diagrams are developed at the beginning of software development to describe the main functions of the software-based system. The use case diagrams describe however only a functionality of software architecture. Therefore we propose to start modeling software architecture from the business context diagram[2] with subsequent process decomposition diagrams[3], and following them with relevant business use case diagrams. The business context diagram proposed by us presents the three dimensions of the software architecture but it shows only key software architecture drivers in one single model, thus is not a big and complex diagram. We show how to relate this context diagram with subsequent process decomposition diagrams, and with the use case diagrams relevant to them.

An early consistency check of the key diagrams is considered important for the consistency and completeness of software architecture, while in principle challenging due to the informal nature of UML specifications. We propose to apply the sufficient criteria for consistent modeling from the business context diagram to the business use case diagrams driven by consistency rules. By sufficient criteria we mean that:

- the context diagram must describe structure, behavior, and functional aspect of the system;
- key elements of the context diagram must be mapped onto process decomposition diagrams;
- key elements of decomposition diagrams must be mapped onto business use case diagrams.

[2] Design diagram based on UML activity diagram describes main business process inputs and output, processing rules and internal structure to store processed data.

[3] Design diagram based on UML activity diagram describes specific business subprocesses decomposed from the main business process.

We propose to apply simple consistency rules to enable automatic verification of consistency of diagrams which present business context, and process decomposition and business use case models. The aim of the consistency analysis is to validate that key elements from those diagrams are mapped onto adequate elements on the conjugated diagrams. Next we propose to verify that each diagram sufficiently describes structure, behavior, and functional aspects of the given system.

The object pseudo-code can be used to formalize this problem and to provide tool support for the analysis. The idea with Z formalization was presented in [NB12] for class, state machine and use case diagrams based on FSB UML diagram. The object pseudo-code may be easily implemented in Java code tools.

Based on our previous work cited above, this chapter presents the sufficient criteria for consistent modeling from the business context diagram to the business use case diagrams driven by consistency rules. In order to apply the proposed criteria we provide a consistency rules for related UML diagrams with the object pseudo-code. We also improve the existing criteria already described and we introduce new rules for the analysis phase. Our concepts are logically extended from the previous papers based on experience of IT projects publicly procured in Poland within the period 2013 – 2014 to the Ministry of Finance and to the Agency for Restructuring and Modernization of Agriculture. It is also underlined here that the proposed criteria driven with consistency rules enable automatic modeling of IT systems.

5.1 Related works

Different software models describe the same system from different points of view, at different levels of abstraction and granularity, possibly in different notations. They may represent the perspectives and goals of different stakeholders. Usually some inconsistencies between models are arising [SZ99]. Inconsistencies in models reveal design problems. If these problems are detected at the early stages of design, costs of fixing them are much lower than dealing with discoveries later during software development or roll-out phases.

Usually UML models are translated into programming languages. Inconsistent UML model may result in an imprecise code. Inconsistencies highlight however conflicts between the views and goals of the stakeholders, thus indicating those aspects of the system which should be further analyzed.

As presented in [HK08], there are several methods to verify consistency in UML diagrams: meta model-based method [PB07], graph-based method [SS 06], scenario-based method, constraint-based methods and knowledge-based methods [WHCZ12]. We are focusing here on constraint-based methods (pseudo-code) and on a graph-based method.

In 2000 Egyed proposed methods for fixing inconsistencies in UML diagrams [E00]. These methods were addressing class, state, object and sequence UML diagrams. The approach to check consistency of activity diagrams was proposed by Jurack et el. in 2008 [JLMT08]. In this method the consistency of the activity diagram was validated with checking whether all flow paths can be performed. Shinkawa in his research [S06] proposed generating consistent UML diagrams from the activity diagram based on Coloured Petri Net. A few rules of consistency between activity diagrams and use case diagrams were proposed by Ibrahim [IISMH11].

Our research studied the consistency of several UML diagrams ranging from the business context diagram (BCM) to the business use case diagrams. We propose the sufficient criteria for consistent modeling of diagrams for context, process decomposition, and business use cases which is driven by the consistency rules. In our method we propose the new consistency rules.

Moreover our approach enables to describe the system in three dimensions i.e. its function, and structure and behavior. E.g. Goel, Rugaber, and Vattam proposed in [GRV09] the structure, behavior, and function modeling language[4]. They viewed SBF as a programming language with specified abstract syntax and static semantics. The SBF language captures the expressive power of the programs and provides a basis for interactive construction of SBF models. They also described an interactive model construction tool called SBFAuthor which is based on the abstract syntax and static semantics of the SBF language. The precise specification potentially provides for a range of additional automated capabilities such as model checking, model simulation, and interactive guides and critics for model construction. The problems of consistency and completeness of models are not however discussed in their paper.

5.2 Criteria of Consistency

According to Functional-Structure-Behavior (FSB) framework introduced by John Gero [G90] the purpose of design description is to transfer sufficient information about target system so it can be constructed. The description must at least provide for function, and structure, and behavior of the target system. Therefore the development of software in which one cannot take into account these three dimensions are "doomed to fail". Truyen [T06] described a model,

[4] SBF hereinafter

in major MDA concepts, as a formal specification of function, structure and behavior of a system. He claims that model must be represented by a combination of UML diagrams. Spanoudakis and Zisman [SZ99] described this as a situation, in which model inconsistencies may arise.

In this section we explain the meaning of consistency of models, which we subsequently apply in the modeling of context, process decomposition, and use case diagrams. Then we present our concept of the sufficient criteria for consistent modeling from BCM (Business Context Model) to BUC (Business Use Case) diagrams.

5.2.1 Model Inconsistencies

To assert that something is consistent firstly we have to declare to what it is consistent with. Software models describe a system from different points of view, at different levels of abstraction with various granularities, and in different notations. Models represent viewpoints, interests and goals of various stakeholders. Such models may easily contradict themselves or in other words loose the consistency. Inconsistencies are usually arising even between diagrams describing the same model.

Inconsistencies reveal serious design problems. Identification of inconsistencies and measures for achieving required consistency can be found in formal methods. The research on consistency models was outlined by Finkelstein [FGHKN94]. Finkelstein stated that inconsistency may be not necessarily a bad thing, and should be evaluated after the translation of the model specification into formal logic. UML is not a formal language so often UML models are translated into more formal notation. Inconsistencies between class, state machine and sequence diagrams in UML [E06] are studied. Inconsistencies arise also because some models are overlapping [SZ99].

UML consistency analysis goes far beyond checking syntax and semantics; it should also encompass other domains like targeted programming language, modeling methodology, modeled systems, and application and implementation domains.

Mens [MSS05] proposed five consistency types:

1. Inter model (vertical) consistency
   Consistency is evaluated between different diagrams and different levels of abstraction. The syntactic and semantic consistencies are also taken into account.
2. Intra model (horizontal) consistency

Consistency is validated between different diagrams but at the same level of abstraction.

3. Evolution consistency
   Consistency is validated between different versions of the same UML diagram.
4. Semantic consistency
   Consistency is validated for the semantic meaning of an UML diagram defined by an UML meta-model.
5. Syntactic consistency
   Consistency might be validated for the specification of UML diagrams in an UML meta-model.

We believe that the Business Context Model should be created firstly and should be already in a consistent form before providing logic for creation of the subsequent models. Such approach prevents from complicated and complex detection of inconsistencies during the model construction.

5.2.2 The Sufficient Criteria for Consistent Modeling from the Business Context Diagram to the Process Decomposition Diagrams

In typical approach of software providers the business context diagrams are not appreciated and often even disregarded as not conveying sufficiently detailed technical information like data formats. Analysts tend to capture the maximum of technical details, put them into weakly structured documents and get quick feedback from the business users (business owners) either directly or using the institution IT staff as go-between. Once comments are received programmers change their cap logos from “ANALYST” to “DESIGNER” to develop business functions directly to get prototype or sometimes even so called “pilot” to be presented to the institution business and IT experts again for feedback. At that moment the communication is usually completely spoiled and the overall project heads towards a disaster at least from the functional capabilities perspective. The business experts got so frustrated with the obvious huge discrepancy between the prototype/pilot and their expectations that the escalation meetings are called whilst significant delays are penalized with the contract clauses. Such situations however might be easily prevented with the disciplined use of context diagrams.

Presumably there is no institution with a full business context model (BCM) developed either for AS IS or for TO BE scenario. A generic model of such BCM diagram is sketched in Figure 5.1.

It represents all meaningful business information flows to and from the institution information system. In practical circumstances of any development (TO BE scenario) and/or maintenance project (both AS IS & TO BE scenarios) a much simpler diagram may be easily initially constructed as shown in Figure 5.2. It represents all business flows to be defined for the project either deducting from the Terms of Reference document and/or from the institution Regulation Book. In both diagrams dependencies describe information flows concerned defining senders and recipients of information elements/objects or signals/events.

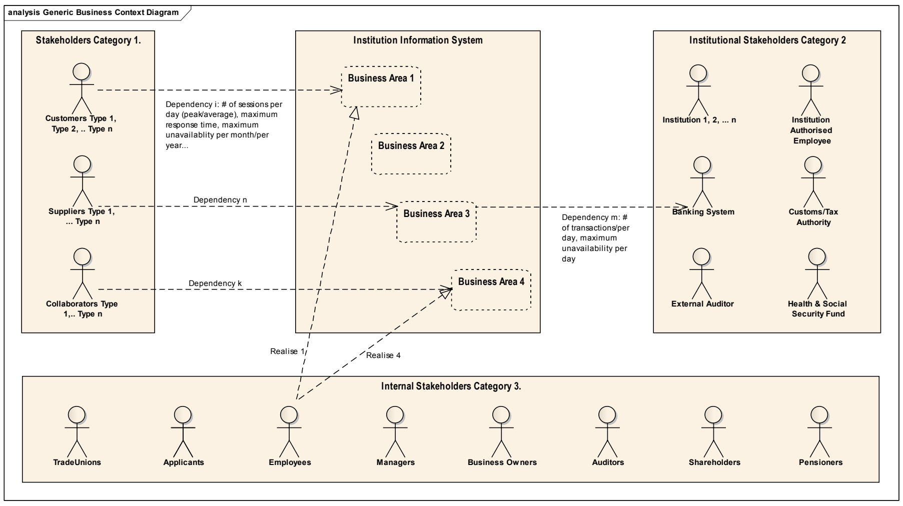

analysis Generic Business Context Diagram
Stakeholders Category 1.
Customers Type 1, Type 2, .. Type n
Suppliers Type 1, ... Type n
Collaborators Type 1,.. Type n
Dependency i: # of sessions per day (peak/average), maximum response time, maximum unavailablity per month/per year...
Dependency n
Dependency k
Institution Information System
Business Area 1
Business Area 2
Business Area 3
Business Area 4
Dependency m: # of transactions/per day, maximum unavailability per day
Institutional Stakeholders Category 2
Institution 1, 2, ... n
Institution Authorised Employee
Banking System
Customs/Tax Authority
External Auditor
Health & Social Security Fund
Realise 1
Realise 4
Internal Stakeholders Category 3.
TradeUnions
Applicants
Employees
Managers
Business Owners
Auditors
Shareholders
Pensioners

Figure 5.1. Generic Business Context Diagram.

Dependencies should also describe all major non-functional requirements like calendar/average and peak densities of data flows, required availability and any other SLA parameters.

The consistency criteria for the business context model start here as follows:

- All flows (dependencies) must be identified (and duly confirmed by business owners of the concerned project[5])

- All types of customers should be identified and categorized due to their different behaviors and numbers[6].

- Each dependency represents at least one and possibly more core business processes initiated in a specific way and delivering specific products (obviously

[5] The best practice recommendation from e.g. PRINCE2 (http://www.prince-officialsite.com)

[6] The principle of BPM in [B01]

there are desirable products but all unwanted products are frequently forgotten). ALL products should be defined with realistic measures[6].

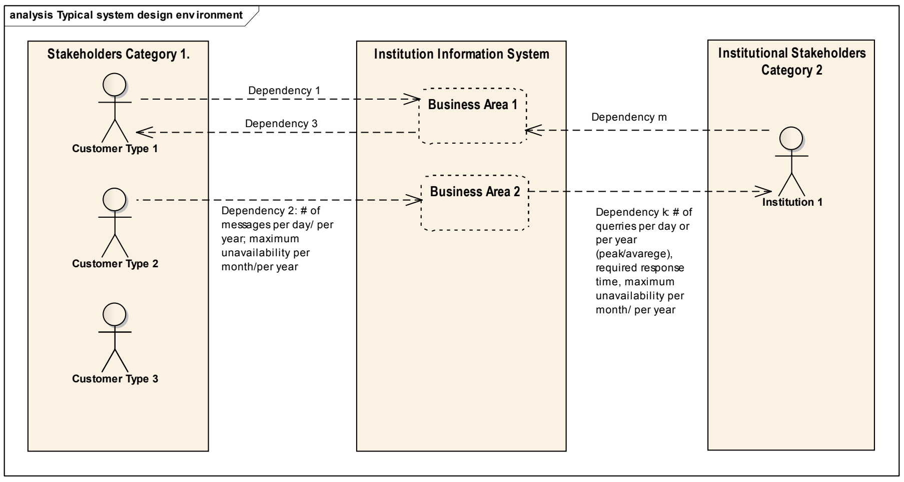

analysis Typical system design environment
Stakeholders Category 1.
Customer Type 1
Customer Type 2
Customer Type 3
Institution Information System
Business Area 1
Business Area 2
Institutional Stakeholders Category 2
Institution 1
Dependency 1
Dependency 3
Dependency m
Dependency 2: # of messages per day/ per year; maximum unavailability per month/per year
Dependency k: # of querries per day or per year (peak/avarege), required response time, maximum unavailability per month/ per year

Figure 5.2. Typical system design environment.

- Each process must be assigned to the specific business owner defining applicable business rules and first of all the strategic goal of the whole process. Once the overall goal is established relevant tactical objectives may be defined. Having goal, objectives and all products defined adequate measures for the process may be proposed and modeled.

Such model of the platform or a system concerned may be drafted and duly agreed with the business owners within 2-3 days. Thus based on agreed well defined reference model there are established proper grounds for further advancement of consistent analytical modeling for any future change requests and even disputes. It is also especially notable that all important factors missing from the initial Terms of Reference are quickly identifiable during that phase and may be agreed at the early stage of contract/development.

We show that all further aspects of the system are traceable to that initial business context model with the described consistency rules.

The business context BCM model should be further decomposed into the following parallel models.

- Actors. Those are all stakeholders identified in the BCM decomposed into distinguishable groups and subgroups depending on their different behaviors and interests to the system.

- Business processes. Those are all flows identified in the BCM multiplied and decomposed if required into more granular behaviors described by activity diagrams. While modeling business processes all major data objects are identified and the structural class model could be created.
- Use Cases. Business processes are subsequently decomposed into use case activity diagrams. Scenarios of Use Cases define the dynamic behavior of the system.

5.3 The Rule-based Method for Consistent Modeling from the Context Diagram to the Use Case Diagrams Driven by Consistency Rules

We propose to start modeling the software architecture from the initial business context diagram with the subsequent process decomposition diagrams, and the relevant business use case diagrams. The verification of consistency of all these diagrams would be obviously needed to identify all omissions and errors in requirements at the early stage of the development process.

5.3.1 Mapping of the business context diagram onto the process decomposition diagram

In Figure 5.3 there are presented rules of mapping the elements from a context diagram onto a process decomposition diagram.

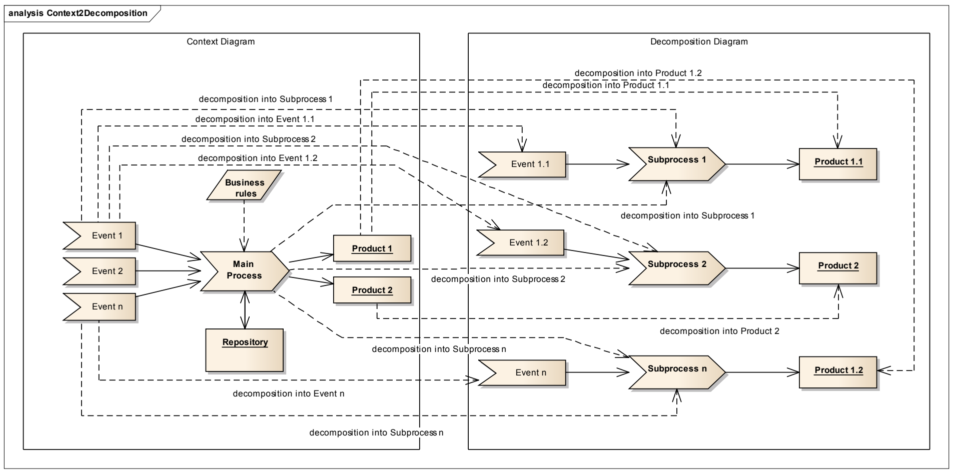

analysis Context2Decomposition
Context Diagram
Decomposition Diagram
decomposition into Product 1.2
decomposition into Product 1.1
decomposition into Subprocess 1
decomposition into Event 1.1
decomposition into Subprocess 2
decomposition into Event 1.2
Business rules
Event 1
Event 2
Event n
Main Process
Product 1
Product 2
Repository
Event 1.1
Subprocess 1
Product 1.1
decomposition into Subprocess 1
Event 1.2
Subprocess 2
Product 2
decomposition into Subprocess 2
decomposition into Product 2
decomposition into Subprocess n
Event n
Subprocess n
Product 1.2
decomposition into Event n
decomposition into Subprocess n

Figure 5.3. Rules for mapping a context diagram onto a process decomposition diagram.

The mapping rules have been described below with a pseudo-code while the following assumptions were made.

1. The context diagram comprises of a single main process, and of at least one event and of at least one product with relevant links/connections between them.
```
public class contextDiagram {
  protected EList<Event> events;
  protected EList<Product> products;
  protected EList<Process> process;
  protected EList<Dependency> dependencies;
}
```
   It should be noted that at the context diagram events describe behaviours, and products describe a structure while processes describe functionality of the modeled system. These are further described as the context diagram sufficient consistency rules.
2. Process decomposition diagram comprises of at least one sub-process, and of at least one event, and of at least one product and of relevant connections/links between them.
```
public class processesDecompositionDiagram {
  protected EList<Event> events;
  protected EList<Product> products;
  protected EList<Process> process;
  protected EList<ObjectFlow> objectflows;
}
```
   It should be noted here that similarly to the business context diagram, at the process decomposition diagram also events describe behaviours, and products describe a structure while processes describe functionality of the modelled system. These are further described as the context diagram sufficient consistency rules

Further on there are rules to transform elements from the BCM diagram onto the process decomposition diagram and then follow diagram validation rules for checking completeness and consistency of diagrams.

Rule 1 (R1) – decomposing of initiating events into initiating sub-events (Event_1 -> Event_1.1 or Event_1 -> Event_1.2):
```
Map<Event> createSubProcesses(contextDiagram, subevents) {
     for(Event event: contextDiagram) {
       for(int i=0;i<=event.getDecomposition; i++) {
         if(event.getDecomposition>0) {
           //new product
             Event subevent =   UMLFactory.eINSTANCE.createEvent();
             //sub-product name
             subevent.setName(event.getName() +
                   event.getSubNumber());
             subevent.setSubprocess(true);//new sub-process
             subevents.add(subevent);        //sub-product list
```

```
            } else
                subevents.add(event);   //product list
          }
        }
        return subevents;
    }
```

Rule 2 (R2) – decomposing of the main process (Level 1) into sub-processes (Level 2) as per initiating events (Event_1, Process -> Subprocess_1 or Event_2, Process -> Subprocess_2 or Event_n, Process -> Subprocess_n):

```
    Map<Process> createSubProcesses(subevents, subprocesses) {
     for(Event event: subevents) {
       if(event.getSubprocess) {
         //new sub-process
        subProcess subproc =
           UMLFactory.eINSTANCE.createSubProcess();
        subproc.setName("Subprocess "+event.getNumber());
        createObjectFlow(subproc, event);
        subprocesses.add(subproc);
      }}
      return subprocesses;
    }
```

Rule 3 (R3) – decomposing of output products into output sub-products (Product_1 -> Product_1.1 or Product_1 -> Product_1.2):

```
    Map<Product> createSubProcesses(contextDiagram, subproducts) {
    for(Product product: contextDiagram) {
     for(int i=0;i<=product.getDecomposition; i++) {
      if(i>0) {
       Product subproduct = UMLFactory.eINSTANCE.createProduct();
       subproduct.setName(product.getName()+product.getSubNumber());
       subproducts.add(subproduct);
       } else
        subproducts.add(product);
     }}
    return subproducts;
    }
```

The above three rules (R1, R2, R3) provide for generating all events, and products and sub-processes for the process decomposition diagram. ObjectFlow type relations could be also generated for events, and sub-events and sub-processes however to complete process decomposition diagram the relations between sub-processes, and products and sub-products should be defined manually.

Rules to map the complete and consistent process decomposition diagram from the complete and consistent context diagram that fulfills the necessary conditions of complete software architecture are as follows:

Rule 4 – there is only one process to represent system behavior;

Rule 5 – there is at least one event influencing the process to represent system function;

Rule 6 – there is at least one product processed by that process to represent system function;

Rule 7 – there is at least one object to represent data structure to be stored within the system exchanging data with the process;

Rule 8 – there is at least one business rule to determine process behavior to represent system behavior.

The necessary completeness rules for context diagrams are stated with pseudo-code below.

Rule 4 (R4) – validation of the existence of a single process:

```
boolean verifyProcessInDiagramContext(contextDiagram) {
     int count=0;
     for(Process proces: contextDiagram) count++;
     If(count==1) return true;
     return false;
}
```

Rule 5 (R5) – validation of the existence of at least one event:

```
boolean verifyEventInDiagramContext(contextDiagram) {
     int count=0;
     for(Event event: contextDiagram) {
             count++;
             if(!contextDiagram.getProcess().getControlFlow()
                 .contains(event.getControlFlow())
                     return false;
     }
     If(count<1) return false;
     return true;
}
```

Rule 6 (R6) – validation of the existence of at least one product:

```
boolean verifyProductInDiagramContext(contextDiagram) {
     int count=0;
     for(Product product: contextDiagram) { //wyszukiwanie produktów
             count++;
             if(!contextDiagram.getProcess().getObjectFlow()
                    .contains(product.getObjectFlow())
                     return false;
     }
     If(count<1)
             return false;
     return true;
}
```

Rule 7 (R7) – validation of the existence of at least one data object:

```
boolean verifyDataInDiagramContext(contextDiagram) {
     int count=0;
     for(Object object: contextDiagram) { //wyszukiwanie obiektów danych
             count++;
             if(!contextDiagram.getProcess().getObjectFlow()
                .contains(object.getObjectFlow())
                     return false;
     }
```

```
        If(count<1)
                return false;
        return true;
}
```

Rule 8 (R8) – validation of the existence of at least one business rule:

```
boolean verifyRuleInDiagramContext(contextDiagram) {
        int count=0;
        for(Rule rule: contextDiagram) { //wyszukiwanie reguły biznesowej
                count++;
                if(!contextDiagram.getProcess().getObjectFlow()
                     .contains(rule.getObjectFlow())
                        return false;
        }
        If(count<1)
                return false;
        return true;
}
```

Once the process decomposition diagram is adequately completed all its elements must fulfil the necessary completeness conditions defined by the following rules:

Rule 9 – each product is an outcome of the process (sub-process);

Rule 10 – each event has to influence the process (sub-process);

Rule 11 – each process (sub-process) has to be initiated with at least one event (sub-event) and has to be completed with at least one output product (sub-product).

Rule 9 (R9) – validation of the consistence of subprocesses and events:

```
boolean verifyProcessesDecomposition(subproducts, subprocesses) {
        for(Product product: subproducts) {
                if(!subprocesses.getObjectFlow.contains(product.getObjectFlow())
                        return false;
        }
        return true;
}
```

Rule 10 (R10) – validation of the consistence of sub-processes and products:

```
boolean verifyProcessesDecomposition (subevents, subprocesses) {
        for(Event event: subevents) {
                if(!subprocesses.getObjectFlow.contains(event.getObjectFlow())
                        return false;
        }
        return true;
}
```

Rule 11 (R11) – validation of the consistence of sub-processes with events and products:

```
boolean verifyProcessesDecomposition (Map<Product> subproducts, Map<Event> subevents,
Map<Process> subprocesses) {
        for(Process process: subprocesses) {
                boolean isEventObjectFlow=false;
                boolean isProductObjectFlow=false;
                for(ObjectFlow objectflow : process.getObjectFlow())
                        if(subevents.getObjectFlow.contains(objectflow) {
                                isEventObjectFlow.set=true;
                                break;}
                for(ObjectFlow objectflow : process.getObjectFlow())
                        if(subproducts.getObjectFlow.contains(objectflow) {
                                isProductObjectFlow=true
                                break;}
                if(!isEventObjectFlow ||!isProductObjectFlow)
                        return false;
        }
        return true;
}
```

5.3.2 Mapping of the process decomposition diagram onto the business use case diagram

The next step to transform the sequence of the business view diagrams (Figure 5.4) is to map elements from the process decomposition diagram onto the business use case diagram.

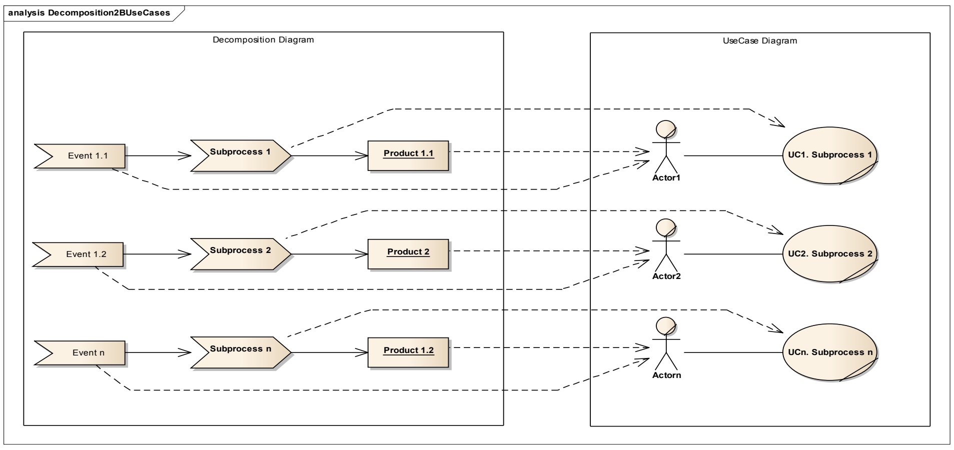


Figure 5.4. Rules of mapping process decomposition diagram onto BUC diagram.

The following assumptions were made for the business process decomposition diagram and for the business use case diagram.

1. The business process decomposition diagram contains at least one business sub-process, at least one event, and at least one product and relevant connections/links between those elements.

```
public class processesDecompositionDiagram {
        protected EList<Event> events;
        protected EList<Product> products;
```

```
          protected EList<Process> process;
          protected EList<ObjectFlow> objectflows;
     }
```

It should be noted that at the context diagram events describe behaviors, and products describe a structure while processes describe functionality of the modeled system.

2. The business use case diagram contains at least one business use case, and at least one actor and relevant associations between those elements.

```
public class businessUCDiagram {
     protected EList<UseCase> usecases;
     protected EList<Actor> actors;
     protected EList<Association> associations;
}
```

It should be noted that at the business use case diagram use cases and actors describe functionality of the modeled system. It should be mentioned also here that UML 2.0 version of standard classifies the use case diagram that supports the business use case diagram as the one of diagrams to describe behavior of a system.

Further on firstly there are rules to transform elements from the process decomposition diagram onto the business use case diagram and then follow diagram validation rules for checking of the created business use case diagram.

Rule 20 (R20) – mapping of business sub-processes onto business use cases (Subprocess_1 -> UC1. Subprocess_1, Subprocess_2 -> UC2. Subprocess_2, Subprocess_n -> UCn. Subprocess_n):

```
Map<UseCase> createBusinessUC(Map<Event> subevents, Map<UseCase> busecases) {
     for(Event event: subevents) {
        UseCase businessUC = UMLFactory.eINSTANCE.createUseCase();
        businessUC.setName("UC"+businessUC.getNewNumber()+event.getName());
        busecases.add(businessUC);
     }
     return busecases;
}
```

Rule 21 (R21) – mapping of input events/subevents and products onto actors (Event_1.1, Product1.1 -> Actor1 or Event_1.2, Product2 -> Actor2 or Event_n, Product1.2 -> ActorN):

```
Map<Actor> createActors(Map<Event> subevents, Map<Actor> actors) {
     for(Event event: subevents) {
        Actor actor = UMLFactory.eINSTANCE.createActor();
        actor.setName(actors.getNewName());
        createAssociation(actor,
           (UseCase) ((Process) event.getSubProcess()).getUC());
        actors.add(actor);
     }
     return actors;
}
```

The above two rules (R20, R21) provide for generating all business use cases, and all actors for the business use case diagram. Associations type relations could be also generated for linking actors and business use cases however to complete creation of the business use case diagram the relations between separate use cases should be defined manually (extend, include).

Once the business use case diagram is properly completed all its elements must fulfill the necessary conditions of completeness and consistency defined with the following rules:

Rule 22 (R22) – validation of consistency of use cases:

```
boolean verifyBusinessUCDiagram(Map<UseCase> busecases, Map<Actor> actors) {
     for(UseCase busecase: busecases) {
             if(!busecase.getAssociation.contains(actors.getAssociation())
                     return false;
     }
     return true;
}
```

Rule 23 (R23) – validation of consistency of actors:

```
boolean verifyBusinessUCDiagram (Map<Actor> actors, Map<UseCase> busecases) {
     for(Actor actor: actors) {
             if(!actor.getAssociation.contains(busecases.get Association())
                     return false;
     }
     return true;
}
```

5.4 Conclusion

In this chapter we have presented a strict procedure to assure consistency of software architecture models in a top-down approach starting with the Business Context Model. The advantages of the approach are as follows.

Developing strict business context model provides for a quick reference for capturing all important non-functional operation aspects while updating and completing system specifications. Change requests are therefore easily traceable and manageable. Business owners and process operators could actively participate in identifying all requirements during the analysis and are aware of its progress thus increasing confidence and improving efficiency of project communications. Our method enables to keep the consistency and satisfactory completeness of the business model.

Finally our method allows for automatic generating of complete business model with very limited need for “manual” programming. In addition, we have shown that the UML diagrams mapped from the business context diagram are consistent.

The next step in our work is to develop the tool automatically generating complete workflow applications based on our method.

References

[B01] Burlton, R.: Business process management: profiting from process, Sams, Indianapolis, IN, 2001.

[CY02] Choi, H., Yeom, K.: An Approach to Software Architecture Evaluation with the 4+1 View Model of Architecture. In: Ninth Asia-Pacific Software Engineering Conference, pp. 286—293. IEEE Computer Society, 2002.

[E00] Egyed A.F., "Heterogeneous View Integration and its Automation," PhD diss., University of Southern California, 2000

[E06] Egyed A.: Instant consistency checking for the UML. In ICSE, pages 381–390, 2006

[FGHKN94] Finkelstein, A., D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh: Inconsistency Handling in Multi-Perspective Specifications. Transactions on Software Engineering, 20(8): 569-578, IEEE Computer Society Press, 1994

[G90] Gero, J. S., Design prototypes: a knowledge representation schema for design, AI Magazine, 11(4): 26-36, 1990

[GRV09] Goel, A., Rugaber, S., Vattam, S.: Structure, behavior & function of complex systems: The SBF modeling language. International Journal of AI in Engineering Design, Analysis and Manufacturing, 23, 23–35, 2009

[HK08] Ha, I., Kang, B.: Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram. In: Fyfe, C., Kim, D., Lee, S.-Y., Yin, H. (eds.) IDEAL 2008. LNCS, vol. 5326, pp. 436–443. Springer, Heidelberg, 2008

[IISMH11] N. Ibrahim, R. Ibrahim, M. Z. Saringat, D. Mansor, & T. Herawan, Definition of Consistency Rules between UML Use Case and Activity Diagram, in T.-h. Kim, H. Adeli, R. J. Robles & M. Balitanas (Eds.), Ubiquitous Computing and Multimedia Applications, ed., Communication of Computer and Information Sciences vol. 151, Springer Berlin / Heidelberg, Daejeon, Korea, 2011

[JLMT08] S. Jurack, L. Lambers, K. Mehner, G. Taentzer, Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams, Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science Volume 5301, pp 341-355, 2008

[K08] Kennaley M.: The 3+1 Views of Architecture (in 3D): An Amplification of the 4+1 Viewpoint Framework. In Seventh Working IEEE/IFIP Conference, pp. 299—302. IEEE Computer Society, 2008

[MSS05] Mens T., Van der Straeten. R., Simmonds, J., A Framework for Managing Consistency of Evolving UML Models, In: Software Evolution with UML and XML, ed. Yang H., chapter 1, 2005

[NB12] Niepostyn Stanisław Jerzy, Bluemke Ilona: The Function-Behaviour-Structure Diagram for Modelling Workflow of Information Systems, w: Advanced Information Systems Engineering Workshops / Bajec Marko, Eder Johann (red.), Lecture Notes in Business Information Processing, vol. 112, Springer, ISBN 978-3-642-31068-3, ss. 425-439 (EOMAS 2012), 2012

[PB07] Paige R.F., Brooke P.J., Metamodel-Based Model Conformance and

Multi-View Consistency Checking, ACM Transactions on Software Engineering and Methodology, Volume 16 Issue 3, July 2007

[S06] Y. Shinkawa, Inter-Model Consistency in UML Based on CPN Formalism, in: 13th Asia Pacific Software Engineering Conference (APSEC '06), pp. 414-418, 2006

[SS06] Shuzhen, Y., Shatz, S.M.: Consistency Checking of UML Dynamic Models Based on Petri Net Techniques. In: Gelbukh, A., Guerra, S.S. (eds.) Proc. of the 15th International Conference on Computing (CIC 2006), pp. 289–297. IEEE Computer Society, Washington, 2006

[SZ99] Spanoudakis, G. and Zisman, A., Inconsistency management in software engineering: survey and open research issues. In: Chang, S.K. (Ed.), Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing Co., Singapore. pp. 329-380, 1999

[TOGAF] The Open Group's Architecture Forum: http://www.opengroup.org/togaf/

[T06] Truyen F., The Fast Guide to Model Driven Architecture, The Basics of Model Driven Architecture, Cephas Consulting Corp, January 2006

[UML] OMG Model Driven Architecture, http://www.omg.org/mda/

[WHCZ12] Wang, Z., He, H., Chen, L., and Zhang, Y., Ontology based semantics checking for UML activity model. Information Technology Journal. 11, 3, 301-306, 2012

Chapter 6

How software development factors influence user satisfaction in meeting business objectives and requirements?

User satisfaction is an useful measure of success of software development projects. The goal of this chapter is to analyze if and how individual factors describing software development process and product are related with selected features of user satisfaction. This chapter investigates two features of user satisfaction: meeting stated objectives (MSO) and meeting business requirements (MBR). Achieving such goal involved using visual techniques as well as a range of statistical and data mining techniques. For MSO there are more identified relationships and they are usually stronger than for MBR. Although there are some disagreements in relationships identified with different techniques, there is a common set of explanatory variables identified by most of techniques. Identified relationships can be used to build more complex simulation or predictive models.

User satisfaction is one of the most important features of software quality. In general, information systems are developed to meet the needs of their users. Satisfaction reflects the level of fulfillment of users' needs by a software system, Thus, the inherent problem with user satisfaction is that it is impossible to express it objectively and difficult to empirically prove what influences it.

Nevertheless, this chapter makes an attempt to identify factors that are related with user satisfaction. Typically, the literature on user satisfaction focuses on application and management perspectives, without links to software development context. In contrast, this study focuses on core software engineering factors.

User satisfaction can be treated as an aggregated measure or broken down into a set of detailed characteristics. This analysis uses the extended ISBSG dataset of software projects [I09] where user satisfaction is expressed by eight variables. This study investigates two of these variables that are important from the business perspective, i.e. user satisfaction with the ability of system to **meet stated objectives** (MSO) and to **meet business requirements** (MBR). Other aspects of satisfaction will be investigated in future studies. The main research questions are as follows:

— RQ1. Which software engineering factors influence MSO and MBR?

— RQ2. What is the nature of these relationships?

To answer these questions, we follow a research approach involving the use of a range of statistical and data-mining techniques (explained in Section 6.2). The main contribution of this chapter is a list of software engineering factors identified by different techniques as related with MSO and MBR. In addition, this chapter discusses the nature of these relationships, pointing out some caution, where appropriate, in interpreting pure quantitative results.

It is important to note that the chapter does not investigate what factors influence if the stated objectives or business requirements are met. Rather, it investigates user satisfaction in these aspects. It is possible that stated objectives are met only in some degree but the user is still generally satisfied with that situation.

6.1 Related work

There are two main related groups of studies, i.e. exploratory studies and prediction studies. The exploratory studies, like the current study, focus on understanding specific phenomena based on analysis of empirical data, expert knowledge, observations, surveys etc. Several such studies investigating user satisfaction have been performed [B05], [MAD12], [P08], [TT10], [WT05]. Studies of user satisfaction with an empirical emphasis have been performed for about 25 years [K93], [W88]. Most of these studies focus on application and management perspectives, typically without strong links to software engineering. In addition, these studies usually involve analysis of very few projects or deeper analysis of just a single project. Thus, while results provided in these studies may be useful in specific context, it is difficult to draw more general conclusions based on them.

In more recent study [SW10] the authors argue that "high rate of developer turnover in projects (due to dissatisfaction) could lead to increasing costs for development firms as well as high user/customer dissatisfaction". The authors observe that with increasing level of user participation the level of developer satisfaction also increases, however the level of user satisfaction slowly decreases. In [RC10] the authors investigate one aspect of user satisfaction, i.e. improving software usability in open-source software. Using a range of statistical methods they analyze factors that might be relevant, i.e., understanding users' requirements, seeking usability experts' opinion by software developers, incremental approach in design, usability testing by managers/developers, and knowledge of user-centered design methods. Since satisfaction with usability will be investigated in future, these results are not relevant for the current study.

The other group of studies aim at building model(s) that could be used to predict future states of certain phenomena based on a set of observations and/or assumed states. While a range of predictive models or frameworks for building them have been proposed in software quality literature [C11], [HB12], [SBH14], very few studies focus on prediction of user satisfaction [FM04], [PP13]. In contrast with the current study, user satisfaction is defined there as an aggregate measure. In [FM04] the focus is on resource prediction; satisfaction depends here on the combination of software quality (mainly its defectiveness) and specification accuracy. A model developed in [PP13] predicts user satisfaction based on the definition of software requirements.

6.2 Methodology and data

This study uses the extended ISBSG dataset of software projects [I09]. The extension means that it contains additional, usually soft, features, such as user satisfaction. Although the ISBSG dataset has been used in numerous studies, e.g. [FG14], [KJ13], [ML08], such extended version in only very few [R11b], [R12]. After numerous data preprocessing steps, explained later in this section, the subset of the dataset used for the main part of analysis contains data on 89 projects described by a set of variables listed in Table 6.1.

The research methodology contains the following steps:

1. **Basic preprocessing**. This step involved activities such as replacing “don’t know” values to “missing”; replacing rare (i.e. with counts close to 1) values of categorical variables by a similar but more common value or marking as “other”; ensuring consistency of values between two variables (e.g. *Client-server* and *Architecture*); removing variables with too many states and very few counts; removing variables with many unclear values (e.g. *Primary programming language*, *1st hardware*, *1st operating system*); transforming variables to their appropriate type (especially logical variables or variables with mixed numeric or interval values); creating logical dummy variables for multi-value categorical variables. This step prepared a dataset to many types of possible future analyses.

Table 6.1. A list of variables used in analysis

Name	Type	N	Notes
Meet stated objectives	logical	89	outcome variable
Meet business requirements	logical	89	outcome variable
Year of project	integer	89	based on project completion date
Adjusted function points	integer	65	also transformed by: ln(x)
Summary work effort	integer	89	in hours; also transformed by: ln(x)
Total defects delivered	integer	72	in the first month after release; also transformed by: ln(x+1)
Development type	nominal	89	new, enhancement, re-development
Architecture	nominal	86	stand-alone, multi-tier/client-server

Name	Type	N	Notes
Client-server	logical	86	
Development platform	nominal	82	PC, Mid-range/multi, mainframe
Language type	nominal	78	3GL, 4GL
Used methodology	logical	81	
Resource level	nominal	89	1-4, the way of recording effort
Debugging tools	logical	72	
Testing tools	logical	72	
Performance monitoring tools	logical	68	
User satisfaction survey	logical	77	
Survey respondent role	nominal	86	customer/user, project manager, sponsor
Project activity scope	logical	77	separate variables for: planning, specification, design, build, test, implement
Organization type	logical	87	separate variables for: computers and software, communications, financial, manufacturing, professional services, other
Application type	logical	88	separate variables for: management information system, network management, transaction/production system
Productivity	numeric	65	in function points per person-hour; also transformed by: ln(x)
Proportion of effort on specification	numeric	67	also transformed by: sqrt(x)
Proportion of effort on build	numeric	81	

2. **Data selection**. This involved filtering the data to the cases with *Data quality rating* set to “A” or “B” as suggested in [I05]; filtering the data so that no attribute describing user satisfaction contains missing values; and filtering the data so that no explanatory variable contains more than 30% of missing data.
3. **Further preprocessing**. This involved repetition of activities as in step 1, but performed on dataset reduced in step 2.
4. **Defining additional variables**. This involved creating variables such as *Productivity*, *Defect rate*, and proportions of effort on specific activities (see Table 6.1). These new variables were then filtered to also meet the criteria for fraction of missing values.

5. **Defining transformed variables**. Since some techniques that were planned to use require normal distribution of variables, this step investigated if such requirement is met and, if necessary, new variables were created after applying commonly used transformations: ln(x), ln(x+1) or sqrt(x) (see Table 6.1).
6. **Defining outcome variables**. The variables describing user satisfaction are originally defined on a 1-4 ranked scale. Since the value "1" and "4" are rare, outcome variables have been defined as logical, i.e. original values "1" and "2" replaced by "false" and original values "3" and "4" replaced by "true" – to indicate if the user satisfaction in particular aspect has been met in a project. Table 6.2 illustrates the distributions for both outcome variables.

Table 6.2. Distributions for two outcome variables

	Meet stated objectives		**Meet business requirements**	
	false	**true**	**false**	**true**
Counts	31	58	19	70

7. **Final variable filtering**. This involved removing variables not suitable for causal analysis, e.g. *Project Id*, *Data quality rating*. Table 6.1 lists variables used in analysis that were kept after this step.
8. **Analysis of correlations for numeric explanatory variables**. This is the first step of the main part of the analysis. Since the outcome variables are dichotomous, this involved the use of a point biserial correlation coefficient which is mathematically equivalent to Pearson product-moment correlation coefficient (assuming encoding values "false"/"true" to "0"/"1") and its interpretation is also the same. Because this step involved using the same statistical test multiple times, i.e. for many explanatory variables, to reduce the risk of reporting false positive results, obtained p values were adjusted according to the false discovery rate (fdr) control [BH95]. This analysis was supported by investigating scatterplots, some of which have been discussed later in this chapter.
9. **Analysis of associations for categorical and logical explanatory variables**. This involved preforming Pearson's chi-squared test or Fisher's exact test for each pair of outcome and categorical/logical explanatory variable. The Pearson's test has been performed when the following conditions were met: a cross-tabulation in a form of z 2x2 table contains at least 5 counts in each cell, in larger tables there are no cells with count of zero and at least 80% of cells have counts of at least 5. Otherwise, the Fisher's test was performed. The p-values obtained in these tests have also been adjusted with "fdr" (as explained in previous step). Statistical-

ly significant relationships, i.e. with p adjusted ≤ 0.05, have been further investigated by analyzing the effect size, i.e. the strength of the relationship, using the following commonly used measures: phi, Cramer's V, Pearson's contingency coefficient, and lambda coefficient.

10. **Analysis of logistic regression models**. The goal of this step was to investigate how each explanatory variable explains the variability of outcome variables. Thus, we built a set of logistic regression models – one for each pair of explanatory and outcome variables. Each of these models contains a single explanatory variable, i.e. it does not take into account any possible interactions between explanatory variables. Such assumption was necessary because adding further variable(s) to the model would result in the need to build the model using fewer cases – most explanatory variables contain missing values and cases with missing values for explanatory variables cannot be used to build the model.
11. **Analysis of data-mining measures of associations**. The goal of this step was to support previous analyses by using other measures of association that are frequently used in data-mining: ReliefF, information gain, gain ratio, and Gini index. The last three measures require variables on non-continuous scale. Thus, to calculate them, each numeric variable has been discretized into five intervals.
12. **Analysis of the CN2 rules** [CN89]. The goal of this step was to learn a set of "if-then" rules that would explain variability in the outcome variables. Such rules can be relatively easily interpreted by human and also used for prediction. What is important, these rules capture the interaction between explanatory variables. An earlier paper [R11b] demonstrated that the CN2 algorithm can produce meaningful and useful rules.

Steps 1-10 were performed with the R statistical software environment [R14] and steps 11-12 using Orange [DCE13].

6.3 Results

6.3.1 Statistical explanatory analysis

The first main step of the analysis was the investigation of relationship between numeric explanatory variables and each outcome variable. Table 6.3 lists the values of point biserial correlation coefficient (r_{pb}) and respective p values (adjusted by "fdr") for each analyzed pair of variables. There are four variables in statistically significant relationship with MSO – one medium-strength positive relationship for the *Year of project* and three medium-strength negative relationships: for *Total defects delivered (ln)*, *Adjusted function points (ln)*, and *Productivity (ln)*.

No statistically significant relationship was found for MBR and any explanatory variable. Without adjusting the p value relationships with *Year of project* and *Adjusted function points (ln)* would be statistically significant.

Table 6.3. Values of measures of association for numeric explanatory variables

Explanatory variable	**N**	**Meet stated objectives**		**Meet business requirements**	
		r_{pb}	**p adj.**	r_{pb}	**p adj.**
Year of project	89	0.40*	0.001	0.25	0.08
Adjusted function points (ln)	65	-0.29*	0.04	-0.28	0.08
Summary work effort (ln)	89	0.14	0.24	0.14	0.34
Total defects delivered (ln)	72	-0.40*	0.002	-0.10	0.59
Productivity (ln)	65	-0.29*	0.04	-0.24	0.13
Prop effort specify (sqrt)	67	-0.19	0.17	0.01	0.94
Prop effort build	81	-0.12	0.28	-0.02	0.94

** - significant at p adjusted. ≤ 0.05*

Figure 6.1 illustrates relationships between strongest numeric explanatory variables and two outcome variables – MSO (parts a and b) and MBR (parts c and d). Parts (a) and (c) confirm that with increasing *Year of project* there are more projects that satisfy both MSO and MBR. However, such straightforward conclusion may be biased by two facts: First, many projects have a *Year of project* with a single common value (2000). Second, there is no project with *Year of project* higher than 2000 for which MSO and MBR were not satisfied. This, based on literature and author's knowledge, cannot be well justified by any theory.

As expected, in projects with fewer defects users were more frequently satisfied in terms of MSO (Figure 6.1, part a). However, no such relationship was found for MBR (Figure 6.1, part c).

Parts (b) and (d) show that with an increased project size (i.e. higher value of *Adjusted function points*) the *Productivity* also increases. Furthermore, in the group of larger projects, the proportion of projects with dissatisfied user increases. However, probably because of imbalanced data, such relationship does not appear to be significant for MBR (Figure 6.1, part d).

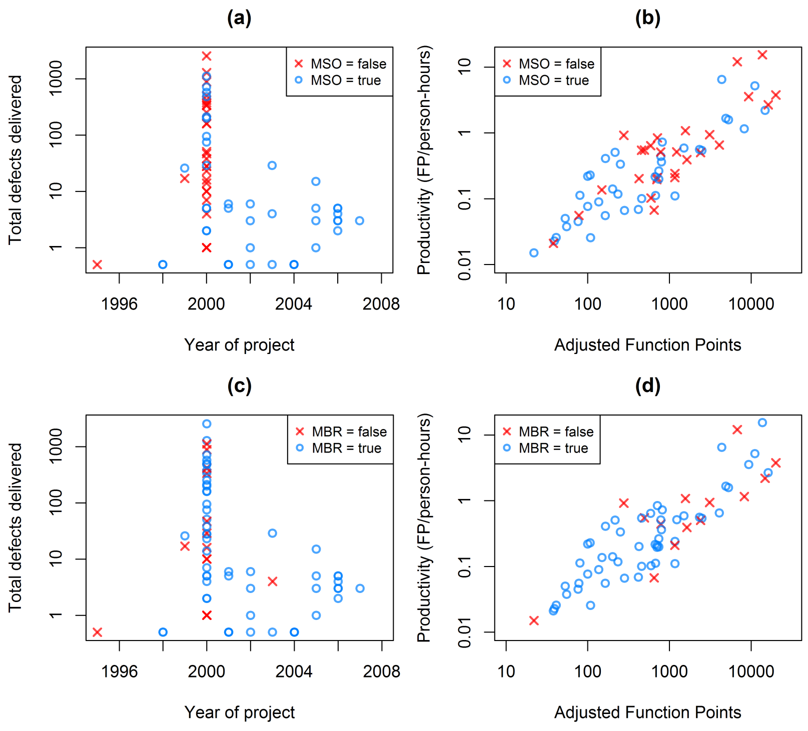


Figure 6.1. Scatterplots for outcome variables and strong explanatory variables

Table 6.4 lists the results of investigating associations between MSO and each explanatory variable. To save space, this table contains only these relationships for which the test of independence confirms the existence of a statistically significant relationship at p adjusted ≤ 0.05. The third column indicates the type of independence test used (either Pearson's chi-squared test or Fisher's exact test), the values of statistic of respective test (where appropriate), and the p value adjusted with "fdr". The last three columns list a range of different measures of effect size. Their values above 0.5 indicate strong relationship, within a range (0.3, 0.5) – moderate, within a range (0.1, 0.3) – weak, and below 0.1 as no relationship. Eight relationships were identified as statistically significant for MSO – at moderate or weak strength. However, in some cases the values of lambda coefficient were unexpectedly low (close to zero) even though other measures of effect size indicate stronger relationship. In these cases, even though there is some level of correlation, these explanatory variables have very low level of ability in predicting MSO. No relationship was found statistically significant here for MBR.

Table 6.4. Values of measures of association for logical and nominal explanatory variables

Outcome	**Explanatory**	**Method Statistic P. adj.**	**Phi/ Cramer's V**	**Pearson's C**	**Lambda**
MSO	Used methodology	Pearson 13.76 <0.01	0.44	0.40	0.41
MSO	User satisfaction survey	Pearson 18.05 <0.01	0.51	0.46	0.47
MSO	Project activity scope design	Fisher ∞ <0.01	0.49	0.44	0.00
MSO	Project activity scope test	Pearson 7.06 0.03	0.33	0.31	0.10
MSO	Application type transaction/ production system	Fisher 4.83 0.03	0.30	0.29	0.00
MSO	Application type management information system	Pearson 8.54 0.02	0.34	0.32	0.24
MSO	Survey respondent role	Fisher – <0.01	0.57	0.50	0.15

6.3.2 Modeling with logistic regression

Table 6.5 provides a list of logistic regression models built for MSO and MBR. Each of these models contains a single explanatory variable and an intercept term (b_0). To save space, this table lists only these models that are statistically significant, i.e. for which p value adjusted with "fdr" is ≤0.05. For MSO ten such models were found statistically significant and only one for MBR. The algorithm for building a logistic regression model provides the values of coefficients in the form of log-odds. To make their interpretation simpler, Table 6.5 provides the values of these coefficients after applying transformation exp(b), i.e. in the form of odd ratios. For example, according to the model, when *Total defects delivered* = 0, then the odds of reaching user satisfaction in MSO is 4.86. This can further be converted to probability of reaching user satisfaction in MSO as 4.86/(1+4.86) = 0.83, which is quite high value – yet, expected when there are no defects. For one unit increase in *Total defects delivered* we expect to see a decrease in odds of reaching MSO to 0.67 of the odds without such additional defect. In general, the value of $\exp(b_0) \in (0,1)$ indicates a multiplicative decrease in the odds of reaching satisfaction in MSO for one unit increase in value for particular explanatory variable, and the value of $\exp(b_0) \in (1,\infty)$ indicates a multiplicative increase in the odds of reaching satisfaction in MSO.

Table 6.5. List of statistically significant logistic regression single-variable models

Outcome	**Explanatory**	**P adj.**	**Exp(b_0)**	**Exp(b_1)**
MSO	Year of project	<0.01	0.00	1.80
MSO	Total defects delivered (ln)	<0.01	4.86	0.67
MSO	Used methodology	<0.01	0.70	7.19 (true)
MSO	User satisfaction survey	<0.01	0.64	11.31 (true)
MSO	Project activity scope - design	<0.01	0.87	1.33e+8 (true)
MSO	Project activity scope - test	<0.01	0.92	4.55 (true)
MSO	Application type - transaction / production system	<0.01	1.22	4.91 (true)
MSO	Application type - management information system	<0.01	3.64	0.23 (true)
MSO	Application type - network management	0.01	1.50	8.00 (true)
MSO	Survey respondent role	<0.01	20.00 (cust./user)	0.03 (project man.) 0.37 (sponsor)
MBR	Survey respondent role	<0.01	1.16e+8 (cust./user)	1.80e-8 (project man.) 4.54e-8 (sponsor)

6.3.3 Rankings by data mining measures

Apart from using “traditional” statistical measures of association discussed in earlier subsections, this analysis also investigated a range of measures that are commonly used in data mining. One of the advantages of these measures is that they can all be applied to any type of explanatory variable – however, for information gain, gain ratio, and Gini index, continuous variables need to be discretized into a set of intervals.

Table 6.6 illustrates the ratings for explanatory variables and each outcome variable using mentioned data mining measures. The higher value of each measure indicates a stronger relationship of particular pair of variables. This is additionally visually indicated by the width of the horizontal bar. The list of explanatory variables has been sorted according to the decreasing order of average strength of relationship calculated as an aggregate for both outcome variables and using all four measures. Thus variables at the top are with the strongest relationship with both MSO and MBR. Due to space constraints this figure contains the top 15 explanatory variables according to this overall rating.

Sorting a list of attributes according to each measure produces different order of explanatory variables for each outcome variable. This is caused by the fact that each measure focuses on different aspect of an association. For example, for MSO the values of ReliefF, information gain and Gini index indicate that *Survey respondent role* is the explanatory variable in the strongest relationship with MSO. However, based on gain ratio, for MSO the strongest relationship is with *Project activity scope: design*. According to the average ranking from all four measures, the top six explanatory variables with the strongest relationship with MSO are: *Survey respondent role*, *User satisfaction survey*, *Project activity scope: design*, *Year of project*, *Application type: management information system*, and *Application type: transaction/production system*. Surprisingly, neither any variable indicating project size nor defectiveness was found as strongly related with MSO.

As for MSO, also for MBR the values of ReliefF, information gain and Gini index indicate that *Survey respondent role* is the strongest related variable with MBR. However, based on gain ratio, for MBR the strongest relationship is with *Organization type: computers & software*. According to the average ranking from all four measures, the top six explanatory variables with the strongest relationship with MBR are: *Survey respondent role*, *Productivity (ln)*, *Year of project*, *Organization type: computers and software*, *User satisfaction survey*, and *Productivity*.

Table 6.6. Ratings for explanatory variables with data mining measures

	MSO				MBR			
Explanatory	**ReliefF**	**Inf. gain**	**Gain ratio**	**Gini index**	**ReliefF**	**Inf. gain**	**Gain ratio**	**Gini index**
Survey respondent role	0.24	0.26	0.17	0.07	0.38	0.14	0.09	0.03
User satisfaction survey	0.21	0.17	0.17	0.05	0.26	0.03	0.03	0.01
Year of project	0.06	0.26	0.14	0.06	0.11	0.08	0.05	0.02
Project activity scope: design	0.18	0.18	0.21	0.04	0.22	0.02	0.03	0.00
Organisation type: computers & software	0.08	0.05	0.08	0.01	0.12	0.06	0.10	0.01
Application type: transaction/production system	0.12	0.07	0.07	0.02	0.22	0.02	0.02	0.00
Organisation type: financial	0.10	0.04	0.08	0.01	0.07	0.02	0.04	0.01
Organisation type: communications	0.07	0.04	0.06	0.01	0.05	0.03	0.05	0.01
Total defects delivered (ln)	0.00	0.09	0.04	0.03	0.13	0.03	0.01	0.01
Adjusted function points (ln)	0.03	0.03	0.01	0.01	0.09	0.06	0.03	0.01
Organisation type: professional services	0.09	0.02	0.03	0.01	0.16	0.02	0.04	0.01
Summary work effort (ln)	0.07	0.02	0.01	0.01	0.16	0.05	0.02	0.01
Application type: network management	0.08	0.05	0.08	0.01	0.08	0.01	0.02	0.00
Organisation type: other	0.12	0.02	0.02	0.01	0.03	0.03	0.04	0.01
Productivity (ln)	0.03	0.01	0.00	0.00	0.11	0.09	0.04	0.02

6.3.4 Modeling with CN2 rules

Before investigation the details of the learnt CN2 rules, let us analyze the performance of the models represented by these rules. This analysis involves a range of measures: accuracy, F1 score, recall, precision, and Matthews correlation coefficient (MCC). While each measure focuses on different aspect, the interpretation of these measures is straightforward: values closer to 1 indicate more accurate prediction while values closer to zero indicate inaccurate prediction. The values of MCC can be negative to indicate predictions opposite to the actual values.

Table 6.7. Performance measures of generated CN2 rules

Outcome	**Validation**	**Acc.**	**F1**	**Recall**	**Prec.**	**MCC**
MSO	test on train data	0.94	0.95	0.91	0.91	0.86
MBR	test on train data	1.00	1.00	1.00	1.00	1.00
MSO	10-fold CV	0.74	0.81	0.83	0.79	0.42
MBR	10-fold CV	0.78	0.87	0.94	0.80	0.15

Table 6.7 provides the values for performance measures of generated CN2 rules. Rules generated for both outcome variables provide very accurate predictions when tested on the same dataset; for MBR they even perfectly explain the relationships. To investigate the adequacy of the CN2 rule generation algorithm we also analyzed performance achieved in 10-fold cross-validation. Naturally, this yielded in lower values for each measure, but still quite high and comparable with using other techniques, such as k-nearest neighbors, classification trees, naïve Bayes or random forests. Thus, this demonstrates the adequacy of CN2 rules. Yet, the predictive aspect of such analysis will be investigated in future studies.

Table 6.8 lists the rules learnt for MSO. We can observe that these rules use explanatory values already identified as related with MSO: *Total defects delivered*, *Adjusted function points* or *User satisfaction survey*. However, some of these rules also use *Summary work effort*, *Prop. of effort on build* or *Development platform*, i.e. explanatory variables that were earlier not identified as related with MSO. This is caused by the fact that earlier techniques were focused on investigating relationships in pairs, i.e. one explanatory variable for one outcome variable. These rules have the ability to capture information on the interactions between explanatory variables. Unfortunately, the quality of some rules is not high (i.e. closer to zero than to one). Furthermore, some rules cover very few projects – even single one at the extreme.

As for MSO, we also generated a set of CN2 rules for MBR (Table 6.9). Here, we can also see a similar set of explanatory variables that are used in these rules. The most important are: *Survey respondent role*, *Used Methodology*, *Summary work effort*, *Adjusted function points*.

Some rules can be easily interpreted and justified causally by the theory of software engineering. For example, a rule "IF Total Defects Delivered>1121 THEN Meet stated objectives=FALSE" means that if there are large number of defects then we should expect not meeting stated objectives. As another example let us analyze a rule "IF Used Methodology=FALSE AND Total Defects Delivered>5 AND Application Type: Network Management=FALSE THEN Meet stated objectives=FALSE". The first two conditions are straightforward – the project does not use any methodology and delivers more than 5 defects (i.e. at least the median of 6 in this dataset). In this case we really should expect no satisfaction in meeting stated objectives. The key is the threshold of tolerable number defects that depends on application type. For network management applications this threshold is typically set to a low value, whereas for other types of application (i.e. perhaps for business use or gaming) it may be higher.

Table 6.8. Rules induced for MSO

Rule quality	**Coverage false/true**	**Rule**
0.29	20:5	IF Used Methodology=FALSE AND Total Defects Delivered>5 AND Application Type Network Management=FALSE THEN Meet stated objectives=FALSE
0.46	6:0	IF Survey respondent role=Project manager AND Summary Work Effort<=898 AND Project Activity Scope Planning=TRUE THEN Meet stated objectives=FALSE
0.55	3:0	IF Adjusted Function Points>457 AND Summary Work Effort>4584 AND Client Server=TRUE AND Summary Work Effort<=9653 THEN Meet stated objectives=FALSE
0.48	1:0	IF Total Defects Delivered>1121 THEN Meet stated objectives=FALSE
0.98	1:0	IF Year of Project>2005 AND Prop Effort Build<=0.00 THEN Meet stated objectives=FALSE
0.20	1:36	IF User satisfaction survey=TRUE AND Summary Work Effort>300 THEN Meet stated objectives=TRUE
0.17	2:9	IF Adjusted Function Points<=250 AND Prop Effort Build>0.00 THEN Meet stated objectives=TRUE
0.26	2:7	IF Adjusted Function Points>649 AND Development Platform=PC AND Summary Work Effort<=4167 THEN Meet stated objectives=TRUE
0.43	1:4	IF Summary Work Effort>2102 AND Total Defects Delivered<=290 AND Adjusted Function Points>649 THEN

Rule quality	Coverage false/true	Rule
		Meet stated objectives=TRUE
0.94	0:2	IF Adjusted Function Points>9296 AND Total Defects Delivered>38 THEN Meet stated objectives=TRUE

However, there are some rules that can be hardly explained by a theory. For example, let us analyze a rule "IF Survey respondent role=Project manager AND Summary Work Effort<=898 AND Project Activity Scope Planning=TRUE THEN Meet stated objectives=FALSE". Over 64% of projects meet the condition for effort. For this rule we cannot find a justification that in these projects when a project involves the planning stage and the survey results are provided by a project manager then we should expect no satisfaction in meeting stated objectives.

Table 6.9. Rules induced for MBR

Rule quality	Coverage false/true	Rule
0.25	6:0	IF Survey respondent role=Project manager AND Used Methodology=TRUE AND Summary Work Effort<=1462 THEN Meet business requirements=FALSE
0.32	5:0	IF Adjusted Function Points>594.00 AND Prop Effort Specify<=0.00 AND Adjusted Function Points>738 THEN Meet business requirements=FALSE
0.34	3:0	IF Productivity>0.00 AND Project Activity Scope Planning=FALSE THEN Meet business requirements=FALSE
0.56	3:0	IF Organization type Other=TRUE AND Application Type Management Information System=FALSE AND Year of Project<=2003 THEN Meet business requirements=FALSE
0.97	2:0	IF Summary Work Effort>9076 AND Summary Work Effort<=9653 THEN Meet business requirements=FALSE
0.09	0:29	IF Total Defects Delivered<=7 AND Summary Work Effort>300 AND Organization type Other=FALSE THEN Meet business requirements=TRUE
0.14	0:18	IF Adjusted Function Points<=781 AND Project Activity

Rule quality	Coverage false/true	Rule
		Scope Test=FALSE AND Adjusted Function Points>22 THEN Meet business requirements=TRUE
0.20	0:10	IF Prop Effort Specify>0.00 AND Summary Work Effort>1779 AND Project Activity Scope Planning=TRUE THEN Meet business requirements=TRUE
0.39	1:10	IF Application Type Transaction/Production System=TRUE AND Summary Work Effort>207 THEN Meet business requirements=TRUE
0.86	0:3	IF Summary Work Effort>5541 AND Application Type Management Information System=TRUE AND Total Defects Delivered<=500 THEN Meet business requirements=TRUE

6.4 Limitations and threats to validity

Results obtained in this study are subject to some limitations and threats to validity. First, the dataset used in analysis is not a random sample from population. Initially, ISBSG gathers data from organizations that are willing to share them. Then, this analysis involved the use of a carefully selected subset of the whole ISBSG dataset, as explained in Section 2. Thus, obtained results cannot be generalized to the whole population of projects.

Second, the dataset contains many missing variables. For this reason many variables have not been used at all, while other still have up to 30% of missing values which (1) may bias the results of single-explanatory-variable models/tests and (2) make it difficult to build multi-variable models.

Next, this analysis involved statistical testing of multiple hypotheses. To mitigate the problem of incorrect reporting inflated number of significant results a false discovery rate control was used to adjust obtained p values.

Furthermore, the analysis of pairs of variables involved pairwise removing cases with missing values. Thus, some relationships cover different projects than other relationships.

Finally, the analysis, especially preprocessing steps, involve subjective decisions, for example on grouping states of categorical variables, setting threshold for fraction of missing values, choosing type of variable transformation, etc.

6.5 Conclusions and future work

Obtained results lead to formulating the following conclusions:

1. Although two investigated outcome variables, *satisfaction in meeting stated objectives* and in *meeting business requirements* seem to describe similar phenomena, there are different explanatory variables in relationships with each outcome variable. I.e. some variables that are related with MSO do not seem to be related with MBR, yet the opposite is less likely.
2. *Survey respondent role* is the explanatory variable with the strongest relationship both for MSO and MBR. To put it simple: it matters mostly who you ask about user satisfaction in terms of meeting stated objectives and business requirements.
3. For *user satisfaction in meeting stated objectives* the strongest relationships are with the following variables individually: *Survey respondent role*, *Year of project*, *Total defects delivered (ln)*, *User satisfaction survey*, *Project activity scope: design*, *Application type: management information system*, *Application type: transaction/production system*, and *Adjusted function points (ln)*.
4. For *user satisfaction in meeting business requirements* the strongest relationships are with the following variables individually: *Survey respondent role*, *Productivity (ln)*, *Year of project*, *Organization type: computers and software*, *User satisfaction survey*, *Adjusted function points*, and *Summary work effort*.
5. Analysis of pure quantitative measures of correlation/association may be misleading and should be supported by other techniques, such as scatterplots, that may reveal issues not directly encoded in specific numeric measure.
6. It is difficult to investigate the relationship of interactions between explanatory variables and outcome variable because of relatively high fraction of missing values in explanatory variables.

In future, this analysis will be extended to answer other important questions related to the impact of software development on user satisfaction. It may involve the use of other features of user satisfaction than selected for this analysis. It may also focus on prediction of user satisfaction based on software development characteristics. For example, the previous papers focused on developing a framework [R11a] for building a Bayesian network model [R13] for software quality simulation and prediction, where user satisfaction is one of many features describing software quality. Such model can be updated by the results of this study.

References

[BH95] Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B,* vol. 57, no. 1, 1995, pp. 289–300.

[B05] Bokhari, Rahat H.: The relationship between system usage and user satisfaction: a meta-analysis. *Journal of Enterprise Information Management*, vol. 18, no. 2, 2005, pp. 211–234.

[C11] Catal C., Software fault prediction: A literature review and current trends. *Expert Systems with Applications*, vol. 38, no. 4, 2011, pp. 4626–4636.

[CN89] Clark P., Niblett T., The CN2 Induction Algorithm, *Machine Learning*, vol. 3, no. 4, 1989, 261-283.

[DCE13] Demšar, J., Curk, T., & Erjavec, A. Orange: Data Mining Toolbox in Python; *Journal of Machine Learning Research*, vol. 14, 2013, 2349−2353.

[FM04] Fenton N., Marsh W., Neil M., Cates P., Forey S., Tailor M., Making Resource Decisions for Software Projects. In: *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC: IEEE Computer Society, 2004, pp. 397–406.

[FG14] Fernández-Diego M., González-Ladrón-de-Guevara F., Potential and limitations of the ISBSG dataset in enhancing software engineering research: A mapping review, *Information and Software Technology*, vol. 56, no. 6, 2014, pp. 527–544.

[HB12] Hall T., Beecham S., Bowes D., Gray D., Counsell S.,A Systematic Literature Review on Fault Prediction Performance in Software Engineering. In: *IEEE Transactions on Software Engineering*, vol. 38, no. 6, 2012, pp. 1276–1304.

[I05] ISBSG, ISBSG Comparative Estimating Tool V4.0 – User Guide, International Software Benchmarking Standards Group, 2005, www.isbsg.org.

[I09] ISBSG Repository Data Release 11, International Software Benchmarking Standards Group, 2009, www.isbsg.org.

[KJ13] Khatibi Bardsiri V., Jawawi D. N., Hashim S. Z., Khatibi E., A PSO-based Model to Increase the Accuracy of Software Development Effort Estimation, *Software Quality Journal*, vol. 21, no. 3, 2013, pp. 501–526.

[K93] Krishnan M.S., Cost, quality and user satisfaction of software products: an empirical analysis. In: *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering*. vol. 1, IBM Press, 1993, pp. 400–411.

[ML08] Mendes E., Lokan C., Replicating studies on cross- vs single-company effort models using the ISBSG Database, *Empirical Software Engineering*, vol. 13, no. 1, 2008, pp. 3-37.

[MAD12] Mitakos T.N., Almaliotis I.K., Demerouti A.G., *What Factors Influence ERP User Satisfaction?: Perceived Usefulness and Self-Efficacy as Key Directors of the ERP User Satisfaction*, LAP Lambert Academic Publishing, 2012.

[P08] Prasad, V.C.S., Complexities in user satisfaction issues during organisational diffusion of in-house developed new technology tools: the case of an Indian IT company. *International Journal of Information Technology and Management*, vol. 7, no. 3, 2008, pp. 315–326.

[PP13] Proynova R., Paech B., Factors influencing user feedback on predicted satisfaction with software systems. In: Doerr, J. ; Opdahl, A. L. (eds.): *Requirements Engineering:*

Foundation for Software Quality, *Lecture Notes in Computer Science*. vol. 7830. Berlin, Heidelberg: Springer, 2013, pp. 96–111.

[R11a] Radliński Ł., A Framework for Integrated Software Quality Prediction using Bayesian Nets, In: *Proceedings of International Conference on Computational Science and Its Applications*, Santander: Springer, 2011, LNCS vol. 6786, pp. 310-325.

[R11b] Radliński Ł., Factors of Software Quality – Analysis of Extended ISBSG Dataset, *Foundations of Computing and Decision Studies*, vol. 36, no. 3-4, 2011, pp. 293-313.

[R12] Radliński Ł., Empirical Analysis of the Impact of Requirements Engineering on Software Quality, In: *Proc. International Working Conference on Requirements Engineering: Foundation for Software Quality*, Essen: Springer, 2012, LNCS vol. 7195, pp. 232-238.

[R13] Radliński Ł., An expert-driven Bayesian network model for simulating and predicting software quality, In: *Proc. Fifth International Conference on Information, Process, and Knowledge Management*, Nice, France, 2013, pp. 26-31.

[RC10] Raza A., Capretz L.F., Ahmed F., Improvement of Open Source Software Usability: An Empirical Evaluation from Developers' Perspective. *Advances in Software Engineering*, vol. 2010, 2010, pp. 1–12.

[R14] R Core Team (2014). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/.

[SBH14] Shepperd M., Bowes D., Hall T, Researcher Bias: The Use of Machine Learning in Software Defect Prediction. *IEEE Transactions on Software Engineering*, vol. 40, no. 6, 2014, 603–616.

[SW10] Subramanyam R., Weisstein F.L., Krishnan M.S., User participation in software development projects. *Communications of the ACM*, vol. 53, no. 3, 2010, p. 137-141.

[TT10] Tarafdar M., Tu Q., Ragu-Nathan T.S., Impact of Technostress on End-User Satisfaction and Performance. *Journal of Management Information Systems*, vol. 27, no. 3, 2010, pp. 303–334.

[W88] White S. D., *An Empirical Study of the User Satisfaction of Accountants with the Software Maintenance Function*, University of Arkansas, 1988.

[WT05] Wixom B.H., Todd P.A., A Theoretical Integration of User Satisfaction and Technology Acceptance. *Information Systems Research*, vol. 16, no. 1, 2005, pp. 85–102.

PART III

SOFTWARE ARCHITECTURE AND DESIGN

Chapter 7

Comparison of selected ESBs on the base of ISO standards

Many IT companies face the problem of software integration. Application of an Enterprise Service Bus (ESB) is one of possible solution to it. There are many ESBs available on the market, differing with basic properties, important from the programmers' perspective. The aim of the chapter is to compare the quality of selected, free ESBs (Mule ESB, JBoss ESB, WSO2). The quality model for tools comparison was built with the use of ISO 9216 standards. The evaluation partially involved an experiment. According to the quality model WSO2 was the best ESB.

Nowadays, software systems written in different programming languages, and run on different platforms, need to communicate each other. The integration problem can be solved in different ways, but the use of Service Oriented Architecture (SOA) seems to be a very beneficial solution [EM1]. The important part of SOA is an Enterprise Service Bus (ESB), a middleware which enables communication with the messages exchange between services.

There are many ESBs available on the market, both paid and free. They differ with their functionality as well as some quality attributes, e.g., performance. The aim of the research presented in this chapter was to compare selected, freely available service buses. For comparison, we chosen the following platforms:

1) Mule ESB 3.4.2
2) JBoss ESB 4.12
3) WSO2 ESB 4.8

The selected solutions are frequently used in practice, and are well documented. Three of the most popular books about ESBs are about chosen systems ("Mule in Action" written by David Dossot and John D'Emic, "JBoss ESB beginner's Guide" by Len DiMaggio and Kevin Conner and "Enterprise Integration with WSO2 ESB" by Prabath Siriwardena). Their popularity has been proved by the number of books sold in Amazon shop and readers in Safari Books Online site.

The evaluation took into account features pointed out as the most important by programmers. This quality perspective is often neglected in similar researches, we managed to find. It must be mentioned that there are not many results of ESB comparisons published in available literature. For example the goal of [INT] was to analyze interoperability of three ESBs (Mule ESB, Fuse ESB, GlassFish ESB) for C4I system. The interoperability was understood as semantic, syntactic, and network interoperability. The winner was Mule ESB which beat the competitors with 1 or 2 points. In [RW24] one can find the comparison of five ESBs (Mule ESB, BEA AquaLogic, Apache ServiceMix, Fiorano, IBM WebSphere ESB) done mainly from functional perspective, with additional elements (price and support). Similarly to the previous research the best solution appeared to be Mule ESB. Comparison of the efficiency of three ESBs (Mule ESB, WS02, and ServiceMix) is the main interest of [SP4]. The comparison was based on experiments implementing 3 different test scenarios. Here, ServiceMix was selected as the best solution followed by WS02.

The quality model used for tools evaluation in our research was proposed on the base of ISO 9126 standard. We decided to use the older standard ISO 9126 not the newer one from the series ISO 25000 because its part ISO 25020, defining the measures, is still unavailable.

7.1 Overview of the selected ESBs

Mule ESB software is being developed by MuleSoft company. The first release was issued in 2005 making it one of the first Enterprise Service Bus platforms. It is characterized by a simple and clear construction. It has two versions: free Community and paid Enterprise version. Creating applications for this system is easier with usage of Mule Studio (based on Eclipse platform) [DD168].

WSO2 first version was developed in 2007. It is one of the few enterprise Service Bus platforms based on OSGi technology. It is characterized by strong support for Active MQ and Axis1. An important function in WSO2 system is a possibility to hot application deploying and support for load balancing [SP86]. It also has installed some security modules, e.g. WS-Addressing, WS-Security, and WS-RM.

First version on JBoss ESB platform was developed in 2006 and was based on Rosetta ESB [DL69]. Same as Mule ESB this platform has two versions free Community and paid Enterprise.

7.2 Comparative procedure

7.2.1 Quality model

Quality model used in our research was based on ISO/IEC 9126 standard [ISO1]. We decide to use this norm instead of ISO 25000 because the older and newer standards have the same comparative model. Newer ISO 25000 has more extensive characteristics and sub characteristics section but in our research norm 9126 is sufficient. This norm focuses on evaluation of software systems. It introduces seven quality characteristics, which in turn consist of sub-characteristics. This standard is very flexible and allows using only selected parts.

To limit the scope of interest to the most important quality characteristics from the users' point of view, we prepared and conducted a questionnaire. The survey was carried out on 30 people from one IT company. The questionnaire was filled by 12 developers, 6 software architects, 5 analytics, 5 testers, and 2 management team members.

Respondents were asked which characteristics are the most important for Enterprise Service Bus. The question in which we ask about importance of quality characteristic was formulated as followed: "Rate (from 1 to 3) the importance of ISO-9126 characteristic where 3 means very important, 2 – important, 1 – unimportant". The results let us to limit further considerations to three quality characteristics (functionality, usability, efficiency) that exceeded the 20% threshold. Characteristics which had less points than 20% of all points have been omitted as less important for ESB comparison.

The weights of quality characteristics, further used in assessment function, were calculated according to the Equation 7.1.

$$w_i = \frac{p_i}{s_i} \tag{7.1}$$

where:

— w_i denotes the weight of i-characteristic
— p_i denotes the sum of points earned by i-characteristic
— s_i denotes the sum of points of three selected characteristic

Table 7.1 presents weights of selected quality characteristics calculated with the previously defined formula (7.1).

Table 7.1. Weights of selected quality characteristics.

Quality characteristic	**Weight**
Functionality	0.43
Usability	0.29
Efficiency	0.28

Every quality characteristic in ISO-9126 standard contains a number of sub-characteristics. Therefore the second aim of the survey was to select only one – the most important – sub-characteristic for every quality characteristic. Respondents were asked to choose one sub-characteristic (the most important from their perspective) for every quality characteristic on the basis of their definitions. In Table 7.2 selected sub-characteristics are gathered.

Table 7.2. Selection of interesting sub-characteristics for quality characteristics.

Characteristic	**Sub-characteristic**
Functionality	Suitability
Usability	Learnability
Efficiency	Time Behavior

Equation 7.2 presents the formula of the final assessment function.

$$F_u = 0.43 * ch_1 + 0.29 * ch_2 + 0.28 * ch_3 \tag{7.2}$$

where
— F_u denotes the value of evaluation function
— ch_1 denotes the result of functionality characteristic assessment (value of suitability metric)
— ch_2 denotes the result of usability characteristic assessment (value of learnability metric)
— ch_3 denotes the result of efficiency characteristic assessment (value of suitability metric)

Each ch_i gains real values between 0 and 1.0.

7.2.2 Assessment of functionality

Suitability was chosen as the most important sub-characteristic of functionality. ISO-9126 standard defines that suitability „determines whether the software has been implemented all the necessary functions required by the user in his daily work. In addition, we include here all the obvious features of the software" [AA5].

Equation 7.3 specifies a metric which is used to calculate functionality. It is defined as the ratio of the number of functions implemented in the system to the expected number of functions [ISO1].

$$X = A/B \tag{7.3}$$

where:

— X denotes the value of functionality assessment
— A denotes the number of functions implemented in the system
— B denotes the expected total number of functions in the system

The following are the properties which Enterprise Service Bus should have according to [RW24]:

— Support for Enterprise Integration Patterns
— Support for Web Services
— Support for HTTP Services
— Support for REST
— Support for SOAP1.1/SOAP1.2
— SOA full platform (ESB, Message Broker, Business Process Server, Data Services Server, Application Server)
— SOA Governance
— Dedicated IDE
— Cloud Connectors

Table 7.3. Assessment of functionality.

ESB Platform	**Number of functions correctly implemented in the system**	**Expected total number of functions in the system**	**Calculation Result**
WSO2	10	10	1
Mule ESB	8	10	0.8
JBoss ESB	9	10	0.9

Table 7.3 shows results for functionality metric calculated for every considered ESB platform. Only one system (WSO2) meets all basic, but most important functions. JBoss ESB and Mule ESB don’t offer the full support for SOA Governance and Mule ESB also doesn’t implement SOA Platform in the whole.

7.2.3 Assessment of usability

The second characteristic selected by respondents was usability. As shown in Table 7.2 the most important sub-characteristic in this case was learnability. ISO-9126 defines learnability as „metrics that assesses how long users take to learn, how to use particular functions, and the effectiveness of help systems and documentation" [AA5].

Equation 7.4 specifies the metric which is used to calculate learnability. It is defined as the ratio of the number of functions described in a tutorial to the expected number of functions that should be described in it [ISO1].

$$X = A/B \tag{7.4}$$

where

- X denotes the value of learnability
- A denotes the number of functions described in a tutorial
- B denotes the expected total number of functions described in a tutorial

The list below indicates the most important basic functions which are very useful at the beginning of the work with Service Buses. This list was prepared based on examples given in [RD38]:

- Short introduction to dedicated IDE
- Deploying simple application
- Using messages flow
- Starting/Stopping server
- System Monitoring
- Creation of simple Proxy application

Table 7.4. Assessment of usability.

ESB Platform	**Number of functions described in tutorial**	**Expected number of functions described in tutorial**	**Calculation Result**
WSO2	4	6	0.67
Mule ESB	4	6	0.67
JBoss ESB	2	6	0.33

Table 7.4 shows results for usability assessment. This characteristic was evaluated by learnability sub-characteristic. It can be noted that none of these three Enterprise Service Buses offers a good tutorial. WSO2 and Mule ESB gained the same amount of points. The JBoss ESB tutorial was the worse. The tutorials were taken from [WS1], [ME1], [JB1].

7.2.4 Assessment of efficiency

Based on the results of the survey, time behavior was chosen as the most appropriate sub-characteristic to study efficiency of each Enterprise Service Bus. To test this property a simple application for each tested system was prepared. The application runs the following scenario (ESB perspective):

1) Receive a message,
2) Add text “hello” to the sent message,
3) Send back the message.

As shown in Figure 7.1, the test system consisted of two parts:

1) Test system – based on JMeter application,
2) Enterprise Service Bus with simple logic implemented (request-response application shown above).

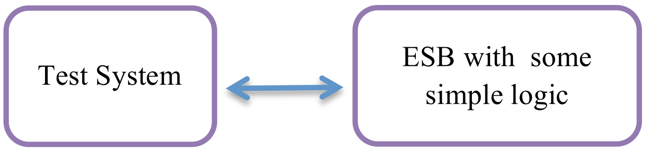


Figure 7.1. System to test time behavior of Enterprise Service Bus

JMeter application responsibility is presented below:

1) Prepare a message,
2) Send a specific number (10, 100, 500, 1000, 3000) of messages concurrently
3) Receive a message and validate it,
4) Generate reports from tests.

The tests were conducted in local testing environment (one node) using the following hardware and software platforms:

— Windows 7 64 bit Professional,
— RAM – 8GB,
— Processor – Intel Core i5.

To calculate time behavior was used Equation 7.5 which is based on the response time metric from ISO – 9126.

$$X = 1 - \frac{p_i}{s}, \qquad (7.5)$$

where

- X denotes the value of efficiency assessment
- p_i denotes the average response time of i-system measured in milliseconds
- s denotes the sum of average response times for all systems measured in milliseconds.

Time behavior was checked for 10, 100, 500, 1000, 3000 massages sent at the same time.

Table 7.5 shows results for efficiency assessment. This characteristic was evaluated by time behavior sub-characteristic. WSO2 gained the highest number of points 0.71. Mule ESB had similar result equal 0.69. System with smallest number of points and thereby with the worst response time was JBoss ESB.

Table 7.5. Assessment of efficiency.

ESB Platform	Average response time	Sum of average response times for all systems	Calculation Result
WSO2	222.8	770.4	0.71
Mule ESB	234.8	770.4	0.69
JBoss ESB	312.8	770.4	0.59

7.2.5 Summary

Final assessment results were calculated with Equation 7.2. Partial scores were shown in tables 7.3, 7.4, 7.5.

Table 7.6. Final results of assessment.

Characteristic	Mule ESB	JBoss ESB	WSO2
Functionality	0.80	0.90	1.00
Usability	0.67	0.33	0.67

Characteristic	**Mule ESB**	**JBoss ESB**	**WSO2**
Efficiency	0.9	0.59	0.71
Final result	**0.3**	**0.67**	**0.82**

System which obtained the largest number of points was WSO2. The final result achieved by this platform was 0.82. In the second place – with a score of 0.73 – was Mule ESB. Third place among selected systems took JBoss ESB with a score of 0.67.

It can be seen that each of the compared platforms has their strengths and weaknesses. Mule ESB has the best help and support to programmers. It also obtained a very good result in performance testing.

JBoss ESB has a lot of features and is adopted to work with many systems. Unfortunately, a disadvantage of this ESB is relative low performance and difficult in reading documentation.

WSO2 platform is the winner in two categories, functionality and efficiency. A little worse result it reached in the usability characteristic.

After analyzing the results it can be stated that during selecting the right Enterprise Service Bus, when functionality, usability and efficiency are important, then a user might be interested in WSO2 platform.

7.3 Conclusions

Integration problems today are often solved with the use of an Enterprise Service Bus. Selection of the proper tool adjusted to specific company goals is not easy. Existing ESBs are rather complex, and differ significantly in the offered features, support and run-time characteristics.

The chapter presents the comparison of three, popular, freely available ESBs: WSO2, Mule ESB, and JBoss ESB. The comparison was done from programmer's perspective. On the basis of the survey conducted among programmers we selected the most important quality characteristics and their sub-characteristics. We limit the scope of our interest to only one sub-characteristic in every category i.e. suitability for functionality assessment, learnability for usability assessment, and time behavior for efficiency assessment. We tried to adapt ISO 9126 measures to assess individual sub-characteristics whenever it was possible, and propose some measures (e.g. Equation 7.5) otherwise. According to proposed quality model (Equation 7.2) WSO2 was selected as the best ESB among three considered. It should be noted that WSO2 was the best solution in all categories (functionality, usability, and efficiency).

In the future we are going to extend the quality model with more metrics and other quality sub-characteristics.

References

[AA5] Alain A. ISO 9126: Analysis of Quality Models and Measures, *Published Online*, 2010

[AN1] Abdullah Alghamdi, Muhammad Nasir, Iftikhar Ahmad, Khalid A. Nafjan, An Interoperability Study of ESB for C4I Systems. *Information Technology (ITSim), 2010 International Symposium in (Volume:2)*, June 2010

[DD168] Dossot D. „Mule in Action. Second Edition". *Manning*, 2012

[DL69] DiMaggio L., "JBoss ESB Begginer's Guide", *Packt Publishing*, 2012

[ISO1] ISO/IEC TR 9126-3 Software Engineering – Product quality – Part 3: Internal metrics, 2002

[RD38] Tijs Rademakers; Jos Dirksen. Open Source ESBs in Action: Example Implemenation in Mule and Service Mix, November 2008

[RW24] Robert Wooley Enterprise Service Bus (ESB) Product Evaluation Comparison. Utah Department Of Technology Services, October 2006

[SP4] Sanjay P, Amit Pattel. Enterprise Service Bus: A performance Evaluation. *Communications and Network*, August 2011

[SP86] Siriwardena, P. "Enterprise Integration with WSO2 ESB", *Packt Publishing*, 2013

[UT20] K. Ueno and M. Tatsubori, "Early Capacity Testing of an Enterprise Service Bus," *IEEE International Conference on Web Services*, Chicago, 18-22 September 2006

[ZC1] G. Ziyaeva, E. Choi and D. Min, "Content-Based Intelligent Routing and Message Processing in Enterprise Service Bus," *International Conference on Convergence and Hybrid Information Technology*, Washington, August 2008

Chapter 8

Architectural patterns applied in Internet of Things

The chapter presents discussion on applying architectural patterns in Internet of Things system class. Internet of Things was partitioned into three application fields: smart home, healthcare and retail. For each of application field there are identified most important architectural drivers and according to them are proposed a few architectural patterns in different architectural views. This chapter discusses issues that each pattern addresses as well as advantages and disadvantages of each pattern in context of Internet of Things.

Internet of Things (IoT) as a vision of a world-wide network of interconnected objects [SS08] is a very promising concept however still in an early stage of development. Most of existing IoT setups are laboratory or experimental size. This is the motivation to deliberate on IoT concept on architectural level, that helps to abstract from implementation matters. Architectural pattern is a standard design or solution in the field of software architecture [BU96]. In general architectural patterns are identified in existing set of software systems, where the similar problem was solved in a similar manner. Architectural patterns are proposed on the basis of classification of problems and solutions that exist in the moment of pattern creation. In case of IoT systems such methodology is not possible as there is not enough experience in IoT deployment. Our research is based on searching for analogies in existing software systems to IoT vision.

In order to document architectural patterns following approach was applied. Patterns are described in context of application fields of IoT. Three application fields has been selected: smart home, healthcare and retail. These three fields represent most beneficial and most promising IoT applications from one side and IoT in these fields seems to be most matured. For each application field there is selected one use case representing IoT. Subsequently two or three architectural patterns in different views/perspectives are enlisted. Each architectural pattern description consists of: pattern context, IoT Domain Model elements instantiated to pattern elements, detailed pattern elements and analogies of pattern application in non-IoT systems.

8.1 State of the art

8.1.1 Internet of Things

Internet of Things as a concept has many definition, among which the most useful is the one proposed by Haller [HA10]: "*a world where physical objects are* *seamlessly integrated into the information network, and where the physical objects can become active participants in business processes. Services are available to interact with these 'smart objects' over the Internet, query and change their state and any information associated with them, taking into account security and privacy issues".* The idea exists for many years (previously known as *ubiquitous computing* or *pervasive computing*) but recently it gained a lot of attention and feasibility due to development in electronic miniaturization, development of the Internet and generally communication protocols. Research on the IoT concept is conducted on many fields [GU13]: communication protocols, energy harvesting and management, security, interoperability, data management and IT architectures. The chpater is focused on IT system architecture supporting IoT.

Comprehensive research in the field of architecture for IoT is presented in [BA13]. The book presents Architecture Reference Model (ARM) for IoT. Important part of ARM is IoT Domain Model [BA11] that is presented in Figure 8.1. Physical objects that are integrated into network are represented by *Physical Entity* class. *Physical Entity* is an abstraction of physical *Device* that consists of tags, actuators and sensors. These objects are represented in information domain by *Virtual Entity*. *Virtual Entity* objects can be passive (referencing to device containing sensors and/or tags) or active (containing actuators). Capabilities of *Virtual Entity* is exposes by *Service*.

Elements of the IoT Domain Model is used in the chapter as base for application architecture patterns.

IoT Domain Model and IoT ARM is based on Service-Oriented Architecture concept. Such approach is widely adopted in architecture considerations on IoT [SP09, TH11].

8.1.2 Architectural patterns

Architectural patterns [AZ05, BU96] are well established and recognized architectural solutions of known problems. Patterns help to document design decisions, improve communication between groups of architecture stakeholders, and offer a common architectural vocabulary.

According to [BU96]: *"Every pattern deals with a specific, recurring problem in the design or implementation of a software system. Patterns can be used to construct software architectures with specific properties."*

Due to above statement patterns are proposed as a result of analyzing previously implemented software systems. Solutions that were successfully used in previously occurred architectural problems are generalized to patterns. This is challenge in applying patterns in case of IoT systems, as there is not much experience in large-scale IoT installations. However in this chapter we are trying to find analogies in non-IoT systems as premises to use them in the IoT field.

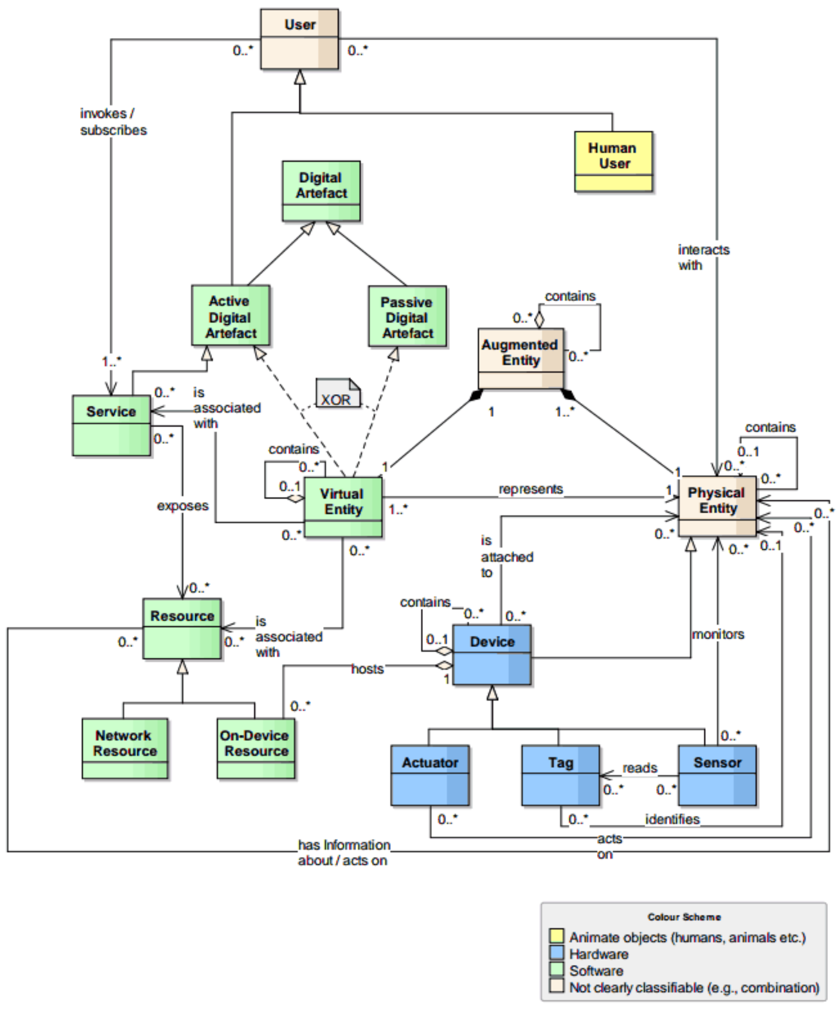


Figure 8.1. IoT Domain Model [BA11].

Architectural patterns are classified based on the concept of architectural views. According to ISO/IEC 42010 [ISO11] architectural view is "*work product expressing the architecture of a system from the perspective of specific system con-*

cerns" where *concern* is *"interest in a system relevant to one or more of its stakeholders. A concern pertains to any influence on a system in its environment, including developmental, technological, business, operational, organizational, political, economic, legal, regulatory, ecological and social influences".*

In this chapter the list of architectural views is based on [BA11] and is as follows:

— functional view
— information view
— deployment and operation view
— evolution and interoperability perspective
— performance and scalability perspective
— trust, security and privacy perspective
— availability and resilience perspective.

8.2 Architectural patterns in smart home application

8.2.1 Use case

Smart home represents idea of connecting objects in a household (lights, air-conditioning, fridge, presence sensors, door/windows opening sensors etc.) in order to improve comfort, decrease power consumption or automate some tasks. Objectives of smart home are similar to home automation but IoT postulate self-organizing of objects, seamless integration and high level of safety and security.

Smart home example scenario is following: a home is equipped with a set of sensors: light, presence, temperature, door/window opened (in each room), smoke and fire detectors, air quality sensor, electricity meters. There are also actuators: light switches, heating, air conditioning switches assigned to individual rooms. All sensors and actuators operate in coordinated manner in order to provide comfort to several residents. Measure of comfort is based on subjective feeling so residents need to provide their feeling (e.g. feeling about temperature or light level).

Another function of smart home infrastructure is monitoring state of the home during absence of all residents and alert on dangerous situations (e.g. fire, flood, burglary) as well as initiate rescue actions: like notify emergency services, turn off electricity (in case of flood) or close gas valve (in case of fire).

IoT smart home infrastructure should also take power consumption into account and operate to minimize consumption without compromising residents' comfort.

Another important requirement is that residents need to monitor/change state of home remotely (via smartphones or web access) but such access has to be safe for security vulnerabilities - non-authorized persons should not be able to access data or modify state.

8.2.2 Implicit invocation (event system) with publish-subscribe, deployment and operation view

8.2.2.1 Context

In the implicit invocation pattern the invocation is not permitted explicitly from calling component to called component, but indirectly through a special mechanism such as publish-subscribe or message queuing or broadcast using additional communicating component. The additional component decouples communicating components. By using the implicit invocation pattern components do not need to be aware of existence or details of other components. Messages that are exchanged between communicating parties are called events.

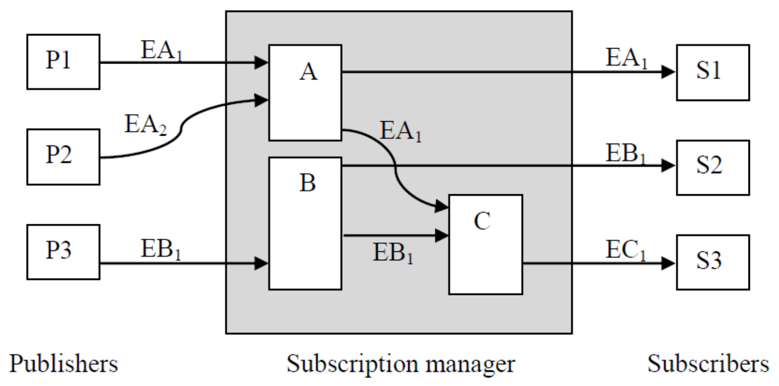


Figure 8.2. Implicit invocations with publish-subscribe mechanism. Publishers generate events *A* and *B* type. Communication hub generates event of *C* type after receiving *A* and *B* type. Subscribers *S1, S2, S3* are subscribed on events *A, B, C* accordingly.

Publish-subscribe mechanism allows event consumers (subscribers) to register for specific events and event producers to publish specific event that reaches a specified set of consumers. Mediating element is called *subscription manager*. Subscription manager is capable of generating new events on the basis of received events (Figure 8.2).

8.2.2.2 IoT Domain Model elements as pattern elements

Application of implicit invocation with publish-subscribe pattern in smart home IoTis as follows. Subscription manager is placed on an element named *hub*. Hub is a hardware and software element that is responsible for communication with all devices attached to all physical entities that establish IoT infrastructure in smart home environment. Hub contains also subscription manager that gathers events, sends events to subscribers and generates new events based on received events. New events are generated according to rules that are stored and processed in rule engine.

Event publishers are:

- tags –identification of physical entity events;
- sensors–change state of environment (temperature, light, door and window is opened or closed, a person enters or leaves room, electricity meter etc.) events;
- services invoked by user–human user feeling expression events (too hot, too cold etc.) or request to modify rule of generating new events.

Event subscribers are:

- actuators;
- services on which are subscribed external users–interface to web application that monitors state of home, alert services, emergency services (police, fire brigade), commercial services (shops) .

Each event consists of following fields: type, name, list of parameters and its values.

New devices that are attached to smart home environment communicate with hub and negotiate automatically role (publisher/subscriber) and type of events that they publish or subscribed to. In order to achieve such functionality there is needed shared directory of events and some semantics of smart home environment (possibly expressed as some kind of ontology) – this aspect is out of scope of the proposed architectural pattern.

During establishing communication between devices and hub there should be provided mechanisms of authorization of devices, but this issue is also out of the scope of pattern.

8.2.2.3 Pattern properties

Application of proposed pattern has significant advantages. The pattern enables simple (from user point of view) attaching new devices to smart home environment. User need only to authorize new device and rest of process of involvement of a device can be performed automatically. Devices do not have to be aware of existence of other devices. Similarly removal of a device can be done automatically.

However this pattern does not provide mechanisms of coordination devices in order to fulfill goals of smart home environment. This functionality can be provided by another pattern, described in the next section – *rule-based system pattern*.

Implicit invocation with publish-subscribe mechanism also does not address problems of self-organizing and security of smart home environment.

8.2.2.4 Analogies in non IoT application

Implicit invocation with publish-subscribe is widely applied in Enterprise Application Integration-in case when many autonomic business applications share information between them in order to support common business goals of an organization.

8.2.3 Rule-based system – functional view

8.2.3.1 Context

A rule-based system pattern is a solution for solving complicated logical problems. It consists of three elements: set of facts, set of rules and engine that process them. Rules represent knowledge in form of a condition and associated actions. Facts represent data. A rule-based system processes each rule that satisfies current condition (based on facts) and executes declared action. The action of a rule might assert new facts, which in turn, triggers other rules.

A rule-base system is an easy and elegant way of expressing complicated logic that is simple to create by a human user (but not always easy to analyze, because of possible large set of simple rules). Rules can be also generated automatically.

8.2.3.2 IoT Domain Model elements as pattern elements

Application of rule-based system pattern is associated with using implicit invocation with publish-subscribe pattern, described in previous section. Facts are events generated by producers and by rule engine.

Rules condition can be any logical combination of event occurrence and event parameter value. Rule action is generation of any event. Generated by rule engine event is send to suitable subscriber. Rule engine is a subscriber of every event.

Set of rules is partitioned into two subsets:

- hard rules –declared only by user or preconfigured, this subset covers rules that defines important actions related with security and safety of home
- soft rules – can be declared by user but also can be generated automatically on the basis of inference, soft rules defines actions concerning comfort and energy saving.

Examples of hard rule:

1. **condition**: occurred event that represents locking door from outside (all residents left house) and occurred event represents window open **action** generate event "alert about opened window"
2. **condition**: occurred event "alert about opened window" and occurred event that represents presence of person in house (possible burglary)
   action: generate event "alert about possible burglary"

On event "alert about possible burglary" should be subscribed sound alarm actuator and service submitting alert to police or security guard.

Examples of soft rules:

1. **condition**: occurred event temperature changed in a room below 20°C and occurred event that represents a resident of the room feel too cold
   action: generate event "turn on heating"
2. **condition**: occurred event a person entered the room and occurred event from light sensor represents change of illumination to dark **action**: generate event "turn on lights in the room"

In case of soft rule no 1 inference engine can gather facts about optimal temperature (optimal in function of residents feeling) and generate rules that will generate events in order to provide comfort. For example turn on heating when temperature decreases below 20°C and turn of when temperature reaches 23°C.

8.2.3.3 Pattern properties

Rule engine performs coordination of devices that conform smart home environment.

Possibility of inference of new rules realizes self-organizing of devices requirement. However this functionality demands utilizing inference engine component with artificial intelligence algorithms.

8.2.3.4 Analogies in non IoT application

Rule-based systems pattern is widely used in expert systems that provides knowledge of an expert or as set of constrains. In case of IoT such experts are home residents that have knowledge about operations of home. Facts can be expressed implicit and put down in a list of rules (hard rules) or gathered during operation of IoT system (soft rules).

8.3 Architectural patterns in healthcare application

8.3.1 Use case

The consequence of the constant population aging is an increasing number of people who require continuous monitoring and medical care. Simultaneously, the possibilities of diagnosis, rehabilitation and pharmaceutical treatment are constantly growing [DA13]. As a result, there are more and more people who are able to live normally despite their diseases. However, they are a group of high-risk - often requiring continuous surveillance - so they are not entirely independent. Healthcare Application, based on the concept of Internet of Things, suggested in [AZ05], could help doctors and other responsible individuals in taking care of patients.

The basic functionality of the application would be to remind the patients to take the medications and to control their absorption. There is a possibility of periodical verification of significant parameters (such as blood pressure or temperature) and also permanent control of vital signs. In case of minor deficiencies noticed, the responsible individual would be informed, in case of large irregularities - alarm would be generated and medical help would be called.

Medical centers and hospitals could accumulate the information from the application, which will significantly facilitate the work of doctors. Basing on real data collected at specific time intervals, they could better assess the progression or regression of the disease. On this basis it will be easier to make decisions about treatment - for example, the need to increase medication dosages. Additional advantage of such solution would be the ability to access data of the patient from other medical centers - if the patient was in another hospital, information about the disease, blood group, allergies, etc. would be acquired instantly.

In this conception it is necessary to use a set of sensors for medical and rehabilitation equipment. The patient should be provided with a unique tag, which identifies him and defines his location. All medicines should be also labeled with tags.

Another essential element is the IoT-enabled smartphone or tablet that patient carries with him, possessing functions such as generating reminders, contacting the responsible individual and calling for medical help.

It is also necessary to communicate with other available IoT systems: for example, in case of overlooking the reminder to take medications, the application could locate patient position and get his attention by pulsed pulsating light (using solutions applied in smart home). In another case, after connecting the application to the sensors on the outside, it could inform the patient with an allergy about high concentration of pollen and suggest an additional dose of anti-allergic, concurrently controlling if the maximum dose is not exceeded.

A particularly important issue of IoT in medical applications is security. Patient's data on his or her condition are sensitive information that should be a subject of special protection. It is of great challenge to concurrently protect the data and share specific parts of the information.

In addition, the application should have a much greater reliability than other IoT applications - it is not difficult to imagine a situation, in which the patient relies on a system that reminds to take medication and this technique fails.

8.3.2 Layered pattern - trust, security and privacy perspective

8.3.2.1 Context

Three-tier architecture is a commonly used pattern in which the communication, functional process logic, computer data storage and data access are developed and maintained as independent modules, most often on separate platforms [EC95]. This solution provides the commonality of resources and ensures availability and efficiency of operations. That type of architecture allows independently updating or replacing modules. In addition, lower and upper layers can be easily designed and implemented using well-known technology (for example, the data storage layer - as databases). The advantage of this solution is also an increase of data security, which are available on the customer request only after passing through an intermediate layer - top and bottom layer cannot communicate directly with each other. A common problem related to this pattern is the complexity of data processing layer - it requires large hardware and computational resources.

8.3.2.2 IoT Domain Model elements as pattern elements

The communication layer includes modules that are responsible for communication with the human users and the physical things (called physical entities) - actuators and sensors. The communication with physical entities can be developed using specific sensor devices, to identify the tags that are used to identify physical entities.

8.3.2.3 Pattern properties

Three-tier architecture seems to be appropriate in perspective of security and privacy of the information in IoT applications related to medicine. Data access is strictly controlled by the intermediate layer, which may apply authorization, authentication and identification mechanisms.

For human users, identification can be performed using special tagged cards (different for patients, doctors, medical assistance), authorization can be performed using passwords entered by the interface. After these processes, intermediate layer would control whether users have the appropriate permissions to the data: medical assistance – only contact details and information on diseases, doctors - information on recent research results, treatment history, etc.

In the case of physical objects, communication may be restricted – it is not allowed to share personal data and other information, which would allow identification of the patient. Only objects which are closely associated with the patient could modify the data - for example pressure readings could only be saved by a pressure gauge equipped with sensor, marked by a unique tag, which would be verified in the intermediate layer.

Communication with unknown sensors could be done while maintaining privacy - for example after detecting sensor for measuring the concentration of pollen in the air, application reads data about the patient’s allergens, and the intermediate layer adds a dozen random allergens - this way information about a specific allergies would be sent undercover.

8.3.3 Broker - evolution and interoperability perspective

8.3.3.1 Context

Architecture for a system that consists of multiple remote objects which interact synchronously or asynchronously and have heterogeneous environment. Need for an easy attachment of new components, which architecture is unknown in this moment and standardize their communications.

8.3.3.2 IoT Domain Model elements as pattern elements

Systems of virtual entities - software representation of devices (actuators, tags and sensors) - able to communicate with the broker.

8.3.3.3 Pattern properties

A broker is a tool that separates the communication functionality of a distributed system from its application functionality. It hides and mediates total communication between the objects or components of a system. The broker is composed of a client-side requestor to construct and forward invocations and a server-side invoker that is responsible for invoking the operations of the target remote object. A marshaller on each side of the communication path handles the transformation of requests and replies from programming-language native data types into a byte array that can be sent over the transmission medium [BA13].

The advantage of the broker, in context of evolution and interoperability of IoT systems, is the easiness of adding new components, even if the architecture is not known. Construction of an application that lists the messages with the environment in a standardized way, allows to exchange the information between systems with potentially very different functionalities, if only subsystems adjust their messages to the broker's standard.

In view of the specific use in medicine, it is easy to imagine a situation, in which solutions used in healthcare applications should be able to communicate with systems in smart home. For example, if sensors used in the smart home want to know the location of the user, they will send a simple query, which is then processed by the broker and sent to healthcare applications in understandable language. The answer is also processed in a way that its result will be understood by the system of smart home.

8.3.4 Redundancy - availability and resilience perspective

8.3.4.1 Context

Systems, in which selected component functionality must be characterized by great reliability. Any type of failure cannot affect fundamental functions of the system, which should continue executing its tasks.

8.3.4.2 IoT Domain Model elements as pattern elements

One of the redundant elements should be components, which analyze the readings from physical entities.

8.3.4.3 Pattern properties

Redundancy is the duplication of critical components of a system with the intention of increasing reliability of the system, usually in the form of a backup or fail-safe [JF76].

The use of redundancy in IoT systems can be understood in two ways. First of them is to duplicate devices responsible for the critical functions of the system - notification of taking medication or calling assistance. In this case, the smartphone of the responsible individual should have the same functions as the smartphone of the patient.

Second way is to duplicate some parts of the control system. Common solution is to triplicate that parts. A defect of one component may then be out-voted by the other two. Simple example of the usefulness of such solution may be duplicating the part responsible for calling emergency - ambulance will be called only when at least two of the three subsystems recognize it as necessary. This will allow an unnecessary call for help to be avoided, which otherwise would be performed in case of failure of a single system.

8.4 Architectural patterns in retail application

8.4.1 Use case

In order to decide if architectural pattern is useful, it needs to be considered on some example. Therefore it was decided to use business case from [BA13] which adhere to logistics, production and sales areas. Key to success for emerging IoT systems is their added value for business. This example very clearly shows potential benefits for companies who will implement this IoT system. Two basic benefits of technology presented in those examples will be reducing damage of the cargo and increasing sales of goods by targeting and profiling each individual customer.

Previously mentioned IoT system spreads at every level of food supply chain. Systems in this business area are extremely complex and distributed, but they are mostly not connected. Basically, making normal, existing systems to be IoT system is about adding sensors at all layers of production and sales (for example at plantation and in market) and connecting all those systems in one organism.

Example IoT system consists of following levels:

- Production – all plants have sensors and RFID readers/tags attached to them. They are generating data about every important factor of environment – humidity, temperature, insulation etc.
- Transportation – each transported product has a sensor. They report data about every event on 24/7 basis. For instance, they can collect data from accelerometer to report if there were any accident during road. Another example could be temperature data that can be crucial in flower transportation. Reaching critical intensity of heat level may lead to death of all living cargo.
- Sales – after the goods are delivered to the market they can have new tags/sensors attached for producing data e.g. about their condition or expiration date. That information are beneficial in management of wares in shop. Additionally, customers can have application on their mobile phones that provides information about availability and position of some wares. Based on data delivered by sensors, profiling service is able to analyze behavior and history of customer's purchases and to present specially tailored discounts.

Thanks to connecting all those parts of system customer can get insight on how the concerned product was transported, in what conditions and what is its origin. Coupling those systems, allowing them to communicate and process data is "making them IoT" and providing vast amount of possible functionalities as well as benefits.

8.4.2 Layers pattern - Functional View

8.4.2.1 Context

This pattern becomes useful in a situation when high-level components need to communicate with lower-level components in order to achieve some goal. Then decoupling those components into different, horizontal layers is beneficial. Each layer has its own interface to allow explicit and loosely coupled communication between each other. In the pure form of the pattern, layers should not be by-passed. Higher-level layers access those in lower-level only through the layer between. It creates a simple and understandable separation. This division helps to maintain modifiability, portability and reusability. This pattern will be most probably used in composition with other ones.

8.4.2.2 IoT Domain Model elements as pattern elements

Figure 8.3 presents layered pattern in production level of exemplary system. Layered pattern is applied for the whole system. The top-down layer represents things from physical world, called physical entities. They can be, e.g., flowers. Next layer consists of devices that are gathering data about those physical entities. In the highest layer are virtual entities - software representations of devices in IoT system.

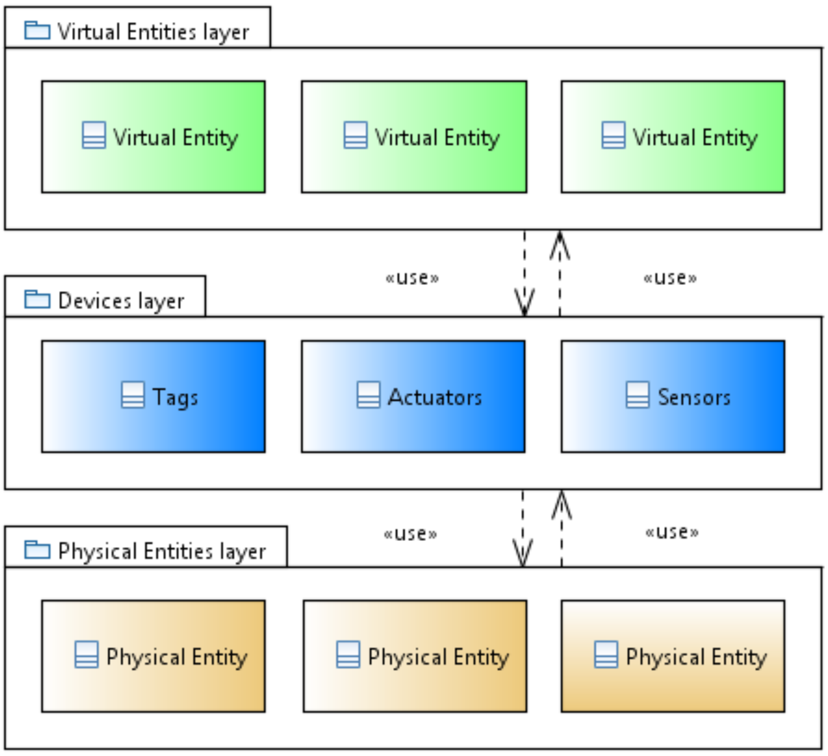


Figure 8.3. Layers pattern in IoT example.

8.4.2.3 Pattern properties

This pattern seems to be natural in IoT. Due to complexity of the domain a clear separation of elements in system will be helpful in giving plain, not over-complicated architecture. Another benefit that can be acquired from using this kind of approach would be easier way to extend any of the layers. Currently, exists some vision of IoT systems, but this particular one can differ or evolve in future due to e.g. emerging new technology. If, for example, new kinds of sensors will appear, then in layered system it would be just a matter of modernizing this one and adjacent layers. First of disadvantages is that data from devices can be acquired only through adequate layer. They should not be by-passed. Another drawback is that any additional layer which needs its own communication interface has impact on efficiency.

8.4.3 Shared repository / Active repository pattern - Information View

8.4.3.1 Context

Another pattern that can be used in given example is shared repository. In this architecture one component of the system is used as a central data store, accessed by all other independent components. It is effective way of sharing big amount of data.

"A shared repository, where all of its clients are independent components, can be considered as client-server, with the data store playing the server part"[AZ05].

Active repository is a variant of this pattern in which clients can be informed about specific events in the shared repository such as changes or access of data. Active repository has a list of subscribers and maintains appropriate communication with them.

8.4.3.2 IoT Domain Model elements as pattern elements

Shared repository store data from virtual entities that are software representation of devices - actuators, tags and sensors.

8.4.3.3 Pattern properties

Concerned business case is data-centered. Systems in this example are constantly gathering and processing data. Appropriate place for storing data is central repository. Later those information can be processed by higher level analyze systems. Stated example needs to have at least three systems – one used in plantation site, another during transportation and last one for sales. To achieve the most benefits and to fulfill idea of IoT, those systems need to communicate with each other. For example, a customer in store wants to check information about potential purchase of flower. She wants to know whether it was ecologically grown, what is its origin and in what conditions it was transported. All those data needs to be gathered from other systems. Those systems collecting data - at production facility, during transportation and in shop can use shared repository architecture. In case of transportation, when alarm functionality is needed, an active repository informing alarm service can be introduced. In holistic architecture of system, data from those shared repositories will be collected, analyzed and used in higher layer what is described in next section.

8.4.4 Enterprise service bus with publish-subscribe - deployment and operation view, evolution and interoperability perspective

8.4.4.1 Context

Enterprise service bus is an additional mediating layer which connects and coordinates exchange of communicates, web services, XML, transformations and data management. It allows usage of service oriented architecture concepts. Its primary use is in enterprise application integration of heterogeneous and complex applications.

8.4.4.2 IoT Domain Model elements as pattern elements

The primary use of enterprise service bus in IoT system will be connecting and allowing communication between active digital artifacts, services and resources.

8.4.4.3 Pattern properties

Enterprise service bus (ESB) brings many benefits. The biggest one is standardized communication that permits many heterogeneous systems to cooperate efficiently. The second is that it allows simple extending of system functionalities. IoT grants a huge amount of possible uses, where many of them still haven't emerged. It is possible to imagine some of them right now, but when the technology will be established widely, many new will flourish. In such situation adding new functionalities to system will be a crucial aspect. That will be satisfied by use of ESB.

The point of IoT is to connect many systems into one. Thus all parts of given example - production, transportation and sales - should be linked. Those parts as described in previous section, can be shared repositories. They are smaller systems that gather data. In stated exemplary organization there is need for "higher layer" system that will connect those repositories, and give "IoT added value". Usage of ESB enables integration of those systems. The amount of different services cooperating only in sales layer is huge as well. For example, customer profiling service, food expiration service, wares availability service and many others. They all need to have a standardized way to cooperate that ESB provides them. In addition, it can be easily imagined that the company is buying another already existing production facility and want to include it in IoT ecosystem. Then if system is fairly extendable, the integration should be smooth and cheap.

Publish-subscribe is useful in e.g. situations when an alarming service is needed. It subscribes for data from signals and if some critical values are exceeded it raises an alarm.

8.4.4.4 Analogies in non IoT application

Every IT system brings challenges to its architecture. Each have its own specific character which can vary depending on type and general purpose. But in all IoT systems main challenges seems to be similar. One of them is scalability, which manifest itself in unknown amount of devices during architecting phase. Also their character is different - they can be heterogeneous. Next challenge is being extendable. Integration of new systems and adding new functionalities should be simple. Those requirements need to be fulfilled and it is possible by using ESB.

8.5 Conclusion

The chapter revised well-established architectural patterns and their applicability in the IoT domain. Important properties of architectures IoT systems where discussed in three different application fields of IoT: smart home, healthcare and retail. Each of application fields has different architectural drivers and should be addressed by appropriate patterns with respect to different architectural views.

Despite the fact that there is not much experience in designing large-scale, IoT domain designers and architects should follow good practices and patterns from IT architecture experience.

References

[AZ05] Paris Avgeriou and Uwe Zdun Architectural Patterns Revisited - A Pattern Language, *In 10th European Conference on Pattern Languages of Programs (EuroPlop 2005), Irsee*, 2005

[BA11] Alessandro Bassi et al. Final Architectural Reference Model for the IoT v3.0 (IoT-A), *www.iot-a.eu*, 2011

[BA13] Alessandro Bassi et al. Enabling Things to Talk, Designing IoT Solutions with the IoT Architectural Reference Model, *SpringerOpen*, 2013

[BU96] Frank Bushman et.al. Pattern-Oriented Software Architecture, *John Wiley & Sons, Inc.*, 1996

[DA13] Timothy M. Dall et al. An aging population and growing disease burden will require a large and specialized health care workforce by 2025, *Health Affairs, 32, no 11*, November 2013

[EC95] Wayne W. Eckerson, Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications.*Open Information Systems*, January 1995

[GU13] Jayavardhana Gubbi et al., Internet of Things: Vision, applications and research challenges*, Future Generation Computer Systems 29 1645–1660*, 2013

[HA10] Stephan Haller, The Things in the Internet of Things, *Proceedings of Internet of Things Conference* 2010

[ISO11] ISO/IEC/IEEE 42010:2011 Systems and software engineering — Architecture Description, *Iso.org.,*2011

[SP09] Patric Spiess et al. SOA-based Integration of the Internet of Things in Enterprise Service, *IEEE International Conference on Web Services, ICWS*, 2009

[SS08] Gérald Santucci and Sebastian Lange Internet of things in 2020 a Roadmap for the Future*, joint EU-EPoSS workshop report*, 2008

[TH11] Teixeira Thiago et al., Service Oriented Middleware for the Internet of Things: A Perspective, *Service Wave 2011, LNCS 6994, pp. 220–229*, 2011

[WA76] J. F. Wakerly, Microcomputer Reliability Improvement Using Triple Modular Redundancy, *Proceedings of the IEEE, VOL.64, No.6*, June 1976

Chapter 9

DCI implementation in C++ and JAVA – case study

The DCI architectural pattern for software, introduced by Reenskaug, contains three parts: Data, Context and Interaction. Data represent domain knowledge while Context and Interaction represent the business logic by implementing communication between objects. Context dynamically injects roles into objects. In strongly dynamic languages like Ruby, the DCI architecture can be easily implemented, in others, like C++ or Java, there are some implementations problems. The goal of this chapter is to present how the DCI architecture can be efficiently implemented in C++ and Java. We briefly describe the DCI pattern and give an exemplary use case implemented in C++ and Java with some explanations.

A design pattern is a general, reusable solution to a commonly occurring problem within a given context in the design of software. It gives a description or a template for how to solve a problem that can be used in many different situations. Patterns are formalized best practices that the programmer must implement in the application [AI97]. They originated as an architectural concept by Christopher Alexander (1977/79) and gained popularity in computer science after the book Design Patterns: Elements of Reusable Object-Oriented Software which was published in 1994 by the so-called "Gang of Four" [GH95]. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application artifacts that are involved. Design patterns reside in the domain of modules and interconnections. At a higher level there are architectural patterns that are larger in scope, usually describing an overall pattern followed by an entire system [GH 95, ST07].

The DCI architectural pattern, introduced by Reenskaug in 2008 [Re08, DC12, DC14], is composed of three parts: Data, Context and Interaction. Data represent the domain knowledge while Context and Interaction represent the business logic by implementing communication between objects. The DCI approach can be used to design lean architecture [CB10] by separating “what-the-system-does” (rapidly changing business-logic features) from “what-the-system-is” (long term domain knowledge). Data represents domain knowledge, Context binds roles to an object; and Interaction represents business logic by implementing communication between roles.

Since its inception in 2008, the DCI approach has spread in many languages, such as Ruby, Python, C++, C#, Java and many more. In strongly dynamic languages like Ruby or Python, a context can dynamically inject roles into an object. The implementation of DCI in these languages is easy and natural. In [BS13, BS14] we gave examples of Ruby code implementing two different use cases, taken from an application which we purposely designed and implemented [ST13] in DCI architecture using Ruby [Ruby] as the implementation language. In literature there is a lack of complete examples showing how to use or implement DCI, only small snippets are published (e.g. in [QZ13]), also simple examples can be found in blogs on DCI e.g. [Mahl], [Ober], [Pars] .

In this chapter we present some implementation problems of DCI in C++ and Java. We illustrate the solutions of these problems through a complete example, different than examples we have presented in [BS13, BS14]. Our contribution is also the explanation how to use and implement the DCI pattern in C++ and Java.

9.1 DCI

While classical object oriented design is mainly focused on classes and objects, the DCI paradigm imposes the view of a system in terms of data, context and interactions [Re08, CR12, Co12, C12a, Re12, Re13, BS13, RC14]. The structure of the system is captured in classes which create objects at the runtime. Classes are tied together by data references and by method invocations that run along those data references. The context refines these relationships on a per-use-case basis with connections that bind together objects in a use case according to their dynamic roles. The context encapsulates the roles that define system behavior and also the bindings between objects and theirs roles. Such view of the system can be easily adapted during design and implementation phases of a new system.

DCI provides a model that allows creating a new set of program structures for each new use case. In DCI classes, are reduced to manage the way the computer represents information in storage. Each use case is implemented in another programming construct called a context which encapsulates behavior defined in a use case scenario. Context also encapsulates the knowledge of how to choose objects and bind their roles to set up dynamic per-use-case relationships. For each use case, appropriate context changes the program structure to create a network of objects cooperating in this use case. The system has a new dynamic architecture for the execution of each use case.

The static aspects in the end user mental model — such as the use case itself — remains static in the DCI code. The DCI model separates the behavioral part from the data structure. The run-time modules are created dynamically according to business needs.

9.2 Examples of DCI

Some widely used programming languages do not fully support DCI paradigm because the dynamic "injection of roles" into object is not allowed in e.g. C++ and Java. Despite this fact, it is also possible to implement DCI based systems in these languages; the examples are given in following sections. We implemented a use case representing an auction. In the presented use case two persons are taking part: a seller and a buyer (bidder). The role of the buyer is to bid, while the roles of the seller are accept or reject the offer. The use cases for the seller and the buyer are shown in Figure 9.1 and Figure 9.2 accordingly.

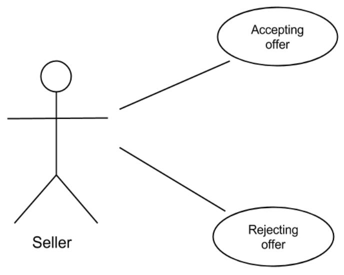


Figure 9.1. Auction - use cases of a seller.

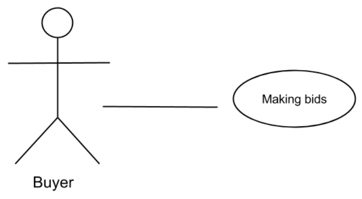


Figure 9.2. Auction - use case of a buyer.

9.2.1 Implementation in C++

Unlike some of the dynamic languages such as Ruby or Python, in C++ objects have to be bound to their roles in the class declaration. In Figure 9.3 the implementation of a use case representing an auction is presented. Roles' identifiers are implemented as abstract C++ classes with their role methods defined as purely virtual, which prevents them from standalone initialization (lines 1-10). They are defined to provide interface for roles and should not implement any logic. Two template classes: `Seller` and `Bidder` are concrete role implementations. The `AuctionContext` class represents auction use case. It bounds roles to their actor objects (lines 72 - 77) and is responsible for the use case enactment. The crucial part of DCI – interaction is defined inside the `execute` method, which triggers the use case execution (lines 84-87).

In C++ implementation of DCI, class has to be injected all roles which can be assigned to any of its instances during the runtime. This results in all role methods being available outside the context, thus violating one of DCI principles. Moreover, it is not possible to call an actor object's method using the role identifier. To address this issue, casting can be used, examples are presented in lines 18, 27, 32 in Figure 9.3.

```
1  class BidderRole {
2  public:
3     virtual void bid(Offer offer, Amount amount) = 0;
4     };
5
6     class SellerRole {
7  public:
8     virtual void accept_offer(Offer offer) = 0;
9     virtual void decline_offer(Offer offer) = 0;
10    };
11
12    #define SELF  static_cast<ConcreteDerived ->(this)
13
14    template <class ConcreteDerived>
15    class Bidder : public BidderRole {
16    public:
17    void bid(Offer offer, Amount amount) {
18       SELF ->name();
19       cout << "bidding...";
20    }
21 };
22
23 template <class ConcreteDerived>
24    class Seller : public SellerRole {
25    public:
26    void accept_offer(Offer offer) {
27       SELF ->name();
```

```
   cout << "accepting...";
 }
 void decline_offer(Offer offer) {
   SELF->name();
   cout << "declining...";
   }
 };


 class User : public Seller<User>, public Bidder<User> {
 public:
 User(){}
 User(string name) {
   name_ = name;
 }
 void set_name(string name) {
   name_ = name;
 }
 void name() {
   cout << name_;
 }
 private:
 string name_;
};

 class Context {
 public:
 Context() {
   parent_ = current_;
   current_ = this;
 }
 virtual ~Context() {
   current_ = parent_;
   }
 public:
 static Context *current_;
 private:
   Context *parent_;
 };

 Context *Context::current_ = NULL;

 class AuctionContext : public Context{
 public:
```

```
AuctionContext(BidderRole* bidder, SellerRole* seller, Offer offer,
 Amount amount_) {
  bidder_ = bidder;
  seller_ = seller;
  offer_ = offer;
  }
BidderRole* bidder() const {
  return bidder_;
  }
 SellerRole* seller() const {
  return seller_;
  }
 void execute() {
  bidder()->bid(offer_, amount_);
  seller()->accept_offer(offer_);
  }
private:
BidderRole* bidder_;
  SellerRole* seller_;
  Offer offer_;
  Amount amount_;
};


User* bidder = new User;
  bidder->set_name("Bidder");
User* seller = new User;
  seller->set_name("Seller");
  Offer offer; Amount amount;

  AuctionContext *context = new AuctionContext(bidder, seller, offer, amount);
context->execute();
```

Figure 9.3. Auction use case implemented in C++.

Other examples of DCI implemented in C++ can be found in [Pars].

9.2.2 Implementation in Java

Java does not allow to dynamically inject code nor does it support multiple inheritance. These features make the DCI implementation in Java difficult. Rickard Öberg and Niclas Hedhman proposed a special tool Qi4j [Qi4j] enabling efficient implementation of DCI in Java using the annotation mechanism. In Figure 9.4 the complete code with the calls of Qi4j library is shown. The code implements the same use case as in Section 9.2.1, i.e., a use case representing an auction.

Both `Seller` and `Buyer` roles are implemented as Java interfaces refined with Qi4j `Mixins` annotations (lines 24-25 and 41-42 in Figure 9.4). Each role interface is provided with a default implementation inside the `Mixin` class. `Mixin` classes define the behaviour specific for a role and hold references to the actor objects (lines 29-38, 47-61). The `Auctioncontext` is a standard Java class, as shown in lines 1 – 22. Context binds roles to their objects upon its creation (lines 9 – 17) and provides one public method `execute`, which is responsible for the interaction between roles.

However the Qi4j library facilitates the implementation of DCI in Java the programmer has to adjust to the requirements imposed by it. The programmer should use the offered by Qi4j skeleton and the dedicated library to access the data. Also the way of running programs is special. In order to run the application, one should use the Qi4j `assembler` classes, which create Qi4j runtime instance as shown in the lines 64 – 77 in Figure 9.4.

```
1     public class AuctionContext
2     {
3     public Bidder bidder;
4     public Seller seller;
5
6     private Offer offer;
7     private BigDecimal amount;
8
9  public AuctionContext(User bidder,
10     User seller,
11     Offer offer,
12     BigDecimal amount) {
13        this.bidder = (Bidder) bidder;
14        his.seller = (Seller) seller;
15        this.offer = offer;
16        this.amount = amount;
17   }
18
19  public void execute() {
20 bidder.bid(offer, amount);
21  seller.acceptOffer(offer);
22   }
```

```

@Mixins(Bidder.Mixin.class)
public interface Bidder{

public void bid(Offer offer, BigDecimal amount);

class Mixin  implements Bidder {

    @This
    User user;

public void bid(Offer offer, BigDecimal amount)
    {
    System.out.println("bidding...");
    }
 }
 }

@Mixins(Seller.Mixin.class)
public interface Seller{

public void acceptOffer(Offer offer);
public void declineOffer(Offer offer);

class Mixin  implements Seller {

    @This
    User user;

public void acceptOffer(Offer offer)
{
    System.out.println("accepting...");
}

public void declineOffer(Offer offer)
{
      System.out.println("declining...");
}
}
}
}
SingletonAssembler assembler = new SingletonAssembler()
{
public void assemble( ModuleAssembly module )
```

```
  throws AssemblyException
  {
module.entities(UserRolemap.class);

  module.services(
    MemoryEntityStoreService.class,
    UuidIdentityGeneratorService.class);
  }
};
  UnitOfWork uow = assembler.module().newUnitOfWork(
  UsecaseBuilder.newUsecase("Auction Context"));

  try {
  User bidder = uow.newEntity(User.class);
  User seller = uow.newEntity(User.class);

  Offer offer = new Offer();

  AuctionContext context = new AuctionContext(bidder,
    seller, offer, new BigDecimal(20));

  context.execute();

  } finally {
    uow.discard();
    }
  }
```

Figure 9.4. Auction use case implemented in Java.

Some problems of DCI implementations with Qi4j are also discussed in R. Oberg' blog [Ober].

9.3 Conclusion

In this chapter we presented a case study on how to implement DCI pattern in C++ and in Java languages. The problems with implementation of DCI paradigm in some programming languages were also noticed by Trygve Reenskaug (the "father" of DCI) and a group of people involved in DCI. They tried to introduce new programming language, called Marvin [Marv]. The syntax of Marvin is based on C# and it fully supports specific for DCI constructs like roles, contexts and dynamic injection of roles into objects. So far we were not able to find any reports about the usage of this language.

DCI is closer to the original goals of the object paradigm in its basis in stakeholder mental models and end user concerns than the regular class-oriented programming. It also preserves the main properties of object-oriented programming i.e.: encapsulation, identity and reflection of human mental models. DCI based systems are very flexible, much more than the traditional ones this is grace to the fact that static (Data) and dynamic (Context, Interaction) part of a system are separated. As each use case is associated with a context it is easy to add new functionalities (use cases) to existing system.

DCI is strictly tied to the system architecture so it seems to us, that it would be very difficult to use it in legacy systems, especially if these systems were designed without the DCI paradigm.

DCI is currently also the subject of several blogs e.g. [Mahl], [Pars], [Ober].

Acknowledgments

We are very grateful to the reviewers for valuable remarks and hints.

References

[AI97] C. Alexander, S. Ishikawa and M. Silverstein. A Pattern Language, New York, Oxford University Press, 1977.

[BS13] I. Bluemke and A. Stepień. DCI pattern. *Software Engineering – selected problems*, Z. Szyjewski, J. Swacha (ed.), Polskie Towarzystwo Informatyczne, ISBN 978-83-7518-598-0, (in polish), 29-39, 2013.

[BS14] I. Bluemke and A. Stepień. Experiences with DCI pattern, submitted to ASC 2014 conference, 2014.

[CB10] J.O. Coplien and G. Bjørnvig. Lean Architecture for Agile Software Development, Wiley, 2010.

[Co12] J.O. Coplien. Reflections on Reflection, *SPLASH'12*, October 19–26, Tucson, Arizona, USA. ACM 978-1-4503-1563-0/12/10, 7-9, 2012.

[CR12] J.O. Coplien and T. Reenskaug. The Data, Context and Interaction Paradigm, *SPLASH'12*, October 19–26, Tucson, Arizona, USA., ACM 978-1-4503-1563-0/12/10, 227, 2012.

[C12a] J.O. Coplien. Objects of the People, By the People, and For the People, *AOSD'12,* March 25–30, Potsdam, Germany, ACM 978-1-4503-1092-5/12/03, 3-4, 2012.

[DC12] T. Reenskaug and J.O. Coplien. The DCI Architecture: A New Vision of Object-Oriented Programming. [online] http://www.artima.com/articles/dci_vision.html, access 2012.

[DC14] DCI – Data Context Interaction. http://fulloo.info/ , [online] access 2014.

[GH95] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Software, Reading, Mass., Addison-Wesley, 1995.

[Mahl] P. Mahlen, blog: DCI Architecture – Good, not Great, or Both, [online] http://pettermahlen.com/2010/09/10/dci-architecture-good-not-great-or-both/

[Marv] Marvin. http://fulloo.info/Examples/Marvin/Introduction/ , access 2013.

[Ober] R. Oberg, blog, [online], http://java.dzone.com/articles/implementing-dci-qi4j

[Pars] Ch. Parson, blog, [online], A fresh take on DCI with C++ (with example), http://chrismdp.com/2012/04/a-fresh-take-on-dci-with-c-plus-plus/

[Qi4j] http://qi4j.org/ , access 2012.

[QZ13] Ch. Qing and Y. Zhong. A Seamless Software Development Approach Using DCI, IEEE 78-1-4673-2008-5/12/, 139- 142, 2012.

[RC14] T. Reenskaug and J.O. Coplien. DCI as a New Foundation for Computer Programming, Programming [online], http://fulloo.info/Documents/CommSense, draft.1.7.pdf, 2014.

[Re08] T. Reenskaug. The Common Sense of Object Orientated Programming, [online], http://folk.uio.no/trygver/2008/commonsense.pdf , Sept. 11, 2008.

[Re12] T. Reenskaug. A DCI Execution Model, [online], http://fulloo.info/Documents/DCIExecutionModel-2.1.pdf , v 2.1, 2012.

[Ruby] http://www.ruby-lang.org/ , access 2013.

[ST07] A. Shalloway and J.R. Trott. Design Patterns Explained: A New Perspective on Object-Oriented Design (2nd Edition), Addison-Wesley, 2007.

PART IV
SOFTWARE QUALITY

Chapter 10

E2A – An Extensible Evolution Analyzer for Software Repositories

Software systems instantly evolve, which is frequently connected with degradation of the code quality. As a consequence, maintenance becomes the essential phase of the software lifecycle. Software repositories record the history of changes in source code that helps to understand the processes of software evolution and propose new methods to effectively manage them. In this chapter we present an extensible software tool for analyzing source code repositories. It allows for defining and collecting metrics, and visualizing their evolution.

Maintenance becomes one of the crucial phases in software lifecycle. Modern systems spend most of their lives being fixed, improved or adjusted. According to the Lehman's laws [Leh96], every software system has to undergo constant adjustment of the code internal structure, or it eventually becomes unmodifiable and unusable. Such a process requires continuous monitoring of parameters vital to the code quality: size, complexity, abstractness etc. In some cases code metrics (like [CK94], [Abr95]) offer a sufficient solution for that, in other – more complex methods and models (like code smells) are required. However, we still need deeper understanding of how the evolution processes impact the metrics and now they can be controlled by refactoring. In all these cases there is a need for specialized, extensible tools that would help developers and in observing and effectively preventing the decay processes in code.

In this chapter we present E2A – a tool that collects data from SVC repositories, conducts analyses on this data and presents their results. Its primary application is observing the software evolution, but it can be also used in software development to analyze trends in quality of the source code. The unique feature of E2A is an API for defining and injecting metrics calculators, which allows users for defining new measures or adjusting the existing ones to their needs. Originally, E2A was designed and developed for Subversion (SVN) repositories. Although numerous recent projects prefer distributed version-control systems (like Git), other long-developed projects still use SVN, which encouraged us to focus on the latter system. However, the layered architecture of the tool allows for accommodating and adapting other version control systems as well.

10.1 Measuring quality of the source code

Code metrics are functions that map a selected feature of the source code to an ordinal scale. However, in order to effectively exploit the information they provide, we need to know the objective context of the measurement, and be able to interpret the results.

The simplest method of introducing the context is comparison with other systems, both in terms of particular values and the trends.. The turning points of a metric values (like its suprema or changes in monotonicity) reveal important data for understanding the software evolution. Additionally, the analysis of several metrics, combined with identifying their inter-dependencies and relations, can frequently display the actual, non-trivial causes for the observed phenomena, that otherwise could be impossible when analyzed in separation.

There are different tools on the market that help collecting, analyzing and visualizing metrics. Plain calculators process the source code and produce values at for various metrics and at different levels (e.g. Eclipse Metrics Plugin [EMP], PMD [PMD]. More sophisticated tools offer the analysis for series of measurements and provide their own statistical capabilities or export data to external systems (e.g. SonarQube [SQ], Alitheia Core [GS09]).

Code smells propose another model of source code quality. They focus not on individual metrics, but on describing symptoms revealing deeper, more complex problems. There are different approaches to identification and detection of code smells. The most popular method is based on *detection strategies,* [Mar04] – quantifiable logical expressions composed of different metrics and calibrated to reflect the perception of a given code smell. This model includes both metrics and their interpretation, and effectively elevates metrics to a higher level of abstraction.

10.2 Concept of the system

10.2.1 Requirements

The initial vision of the tool presented in the previous section, has been transformed into a set of specific requirements. We chose user stories as a method for describing and managing them. User stories are compact, user-centric, and allow for capturing various points of view [Bec99]. A story is a structured narrative sentence, reflecting a *role* (who typically describes the initiator and beneficiary of the story), a *desire* (describing the functionality required by the user) and the *benefit* (which reflects the rationale for the function). For E2A, there is only one role: a user.

Package A: Metrics management

A1: Selection of metrics

As a user of the system, I want to be selecting metrics to be calculated.

A2: Selection of the granularity level

As a user of the system, I want to calculate metrics values at different granularity levels: package, class and method.

A3: Analysis of a branch in the repository

As a user of the system, I want to analyze a selected branch in the repository and analyze the evolution of a particular release.

A4: Comparative analysis of branches in the repository

As a user of the system, I want to compare the evolution of metrics values on selected development branches in order to observe how the selected qualitative characteristics changed with time.

A5: Adding new metrics definitions

As a user of the system, I want to add new metrics definitions without changing and recompiling the existing system, in order to easily extend capabilities of the system.

Package B: Visualization

B1: Visualization of metrics

As a user of the system, I want to visualize metrics for selected elements of the system, in order to easily analyze the dynamics of the change and relations between metrics. It is essential to present several metrics simultaneously.

B2: Identification of classes of particular interest

As a user of the system, I want to process and aggregate values of the metrics in all classes/packages, in order to identify classes that require further analysis.

B3: Reporting

As a user of the system, I want to be able generating report on my current analysis in a format accepted by MS Excel, in order to continue the analysis outside the system.

Package C: Storage

C1: Caching and storing the data

As a user of the system, I want to retrieve data from repository only once and calculate the metrics several times without a need to reconnect, in order to reduce time required to analyze the projects. Before closing the system the current analysis should be stored.

10.2.2 Architecture

The tool is composed of two main modules: (i) *a code analysis module* and (ii) *a graphical UI.*

The code analysis module has layered architecture, in which every layer depends on the layer below, and their interactions are described by well-defined contracts. Such structure enables replacing selected layers without affecting the remaining parts of the system, e.g. changing the library responsible for handling the access to the repository would not require changes in the code analysis components. Specifically, there are 4 layers in that module:

- *repository access*, responsible for running all file-based operations on the repository; it also encapsulates the SVC protocol being used;
- *compilation units,* responsible for grouping files (transformed to compilation units) in revisions or development branches;
- *Abstract Syntax Tree (AST)*, which generates syntax trees for the selected compilation units;
- *metrics calculators* that operates on ASTs generated from selected revisions or branches.

In order to additionally separate the layers and facilitate declarative dependency resolution, the communication between them is based on the Dependency Injection pattern.

The UI module has been implemented as an Eclipse plugin, in order to take advantage of several useful services offered by this platform or available as external components (in particular, JDT AST and metrics calculators). The UI module is tightly coupled to all layers within the code analysis module. This appeared necessary in order to exploit specific features of the accessed SVC (like Subversion or Git).

10.2.3 Overview of the analysis process

In general, the tool offers two operational modes:

- analysis of revisions in a selected branch,
- comparative analysis of several branches.

In both modes, *a revision* is the basic notion. It is the snapshot of state of repository at particular time. Other elements of the tool, like compilation units or metrics values, are associated with a given revision. In case of the comparative analysis of several branches, each of them is also analyzed as a revision.

The analysis starts with examining history of the selected branch. As a result, a list of changes in subsequent revisions is generated, which is further used for reconstructing the source files. Two options are available:

- querying the repository for a single source file content in a selected revision (which is costly, especially for remote repositories),
- incremental retrieval of subsequent revisions of the source file into a local working copy in a temporary folder (much more efficient).

At that point the lists with changes in particular version, along with the source files, is delivered to the code analysis module. They are compiled and transformed to the compilation units; additionally, the AST is generated, which is required for metrics calculation. The latter ones are organized into code graphs, stored separately for every revision. Noticeably, the code graph is constructed incrementally, whereas its snapshots are stored. Additionally, these snapshots can be saved to a file and retrieved later, which helps the user in managing the analysis process. Next, the code graphs, containing compilation units, are sorted by version. In that form the data is made available to the metrics calculators. There are two types of metrics: *base measures* and *derive measures*. The base measures just count occurrences of a given phenomenon, while the derive measures collect results from a set of compilation units and aggregates them according to a selected strategy: an average, a maximum, a minimum or a sum of particular values.

Due to the adopted iterative method of processing repository data, the main limitation of the tool is the need for analyzing the entire selected branch, starting from the first revision. It is particularly visible in case of multi-projects repositories, which often contain a large number of revisions. Moreover, public repositories (e.g. Google Code) often impose a limit of queries a user can issue in a time period. To address this issue, E2A checkouts a working copy of the analyzed project, which allows for using local queries instead of remote ones. The increased consumption of the disk space in return seems an acceptable price for that.

10.3 Example of use

In order to verify the capabilities of the tool on a real repository, we analyzed PowerMock – a FLOSS project hosted at Google Code[1]. It is a mock object library meant to facilitate unit testing by instrumenting binary Java code and generating mock objects and methods at runtime. The project has ca.1800

1 https://code.google.com/p/powermock

revisions in repository, and its development dates back to 2008. Since then it has had 29 releases (tagged in repository). We analyzed ca. 400 different compilation units at two levels of granularity: revisions and releases.

There are three main modules that comprise the project: *core*, *modules*, *reflect*. In release analysis, we analyzed them separately (except for the programming examples stored in repository and the API for other libraries). Due to space limitations, we present only some of the problems and solutions. In revision analysis, we focused on the main branch (*trunk*).

10.3.1 Analysis of releases

The analysis of the project's health starts with an overview of the average cyclomatic complexity for entire system. Figure 10.1 presents how the system size changed between releases 0.5 and 1.5. System was growing steadily, except for the releases 1.0-1.3, when we observe two sudden size outbursts, separated by a stability period, in which particular classes were relatively large and contained numerous methods. Later, the relation between code size and the method number became moderate, which probably was caused by performing refactoring that balanced the responsibility of individual classes.

Figure 10.2 presents again the system complexity, supplemented with data of the number of types in the system (red), average depth of inheritance (green) and total number of subclasses in the system (violet). The growth of the inheritance metrics values, accompanied by the greatest drop of complexity (marked with a dashed line), may suggest intensive refactoring activities that improved the reuse of superclasses. In this case, extending the depth of inheritance tree (i.e. extracting intermediate classes) could be used for reducing complexity of the implementation classes in the lower part of the tree.

In order to identify the module responsible for complexity variation, we analyzed the complexity of individual components (*reflect*, *modules* and *code*), depicted at Figure 10.3. It turned out that the *reflect* package is mostly responsible for the overall increase in that metric (the average complexity appears significantly higher than for the entire system).
Moreover, the complexity was moderated simultaneously with the inheritance changes in the component, which seems to confirm the aforementioned hypothesis. However, as presented at Figure 10.4, the *modules* component demonstrates a different dynamics. In the previously mentioned period (releases 1.0-1.3), we observe only a small growth of complexity for that component, and the changes in inheritance hierarchy appear in recent releases. Effectively, this

component is developed in a more balanced way, with a moderate growth of complexity.

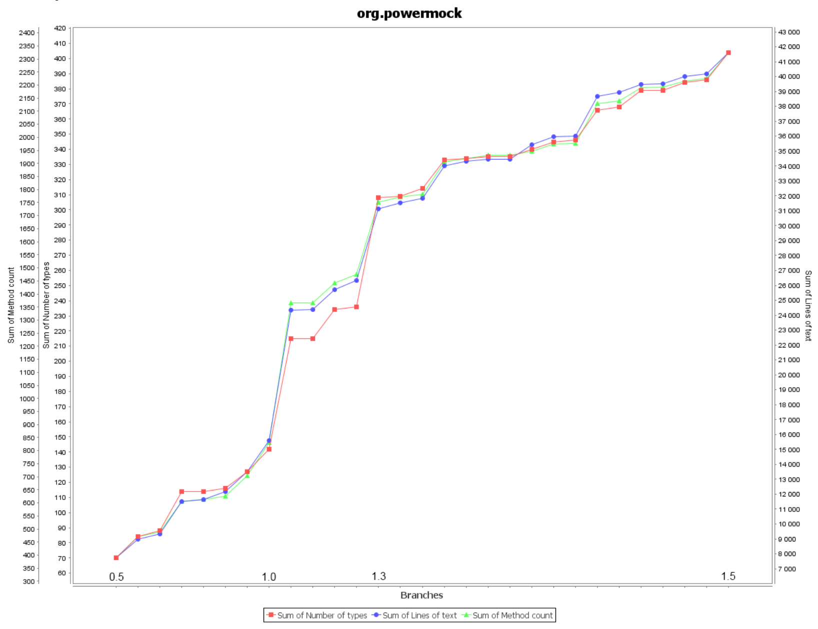


Figure 10.1. System size (#types, #LOC, #methods), by release.

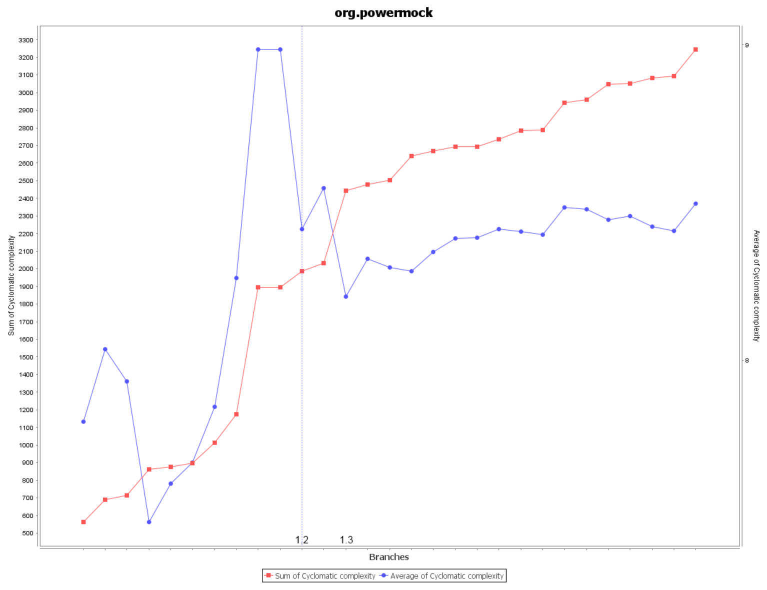


Figure 10.2. System complexity, by release.

10.3.2 Revision analysis within a branch

The revision analysis was conducted on subsequent committed revisions of the main development branch (*trunk*). Unlike the previous analysis, this one ignores the release tags and presents the data for all revisions.

The results of the complexity measurement are presented in Figure 10.5. We can identify the revision that dramatically increased the total LOC and CC metrics for entire project. After referring to the source code, we identified a huge refactoring that was applied at that time (that was also suggested by the commit comment). Next, we analyzed the impact of the refactoring on the average complexity of the system. It increased the complexity and also increased the differences between classes. Later, despite the growth of complexity for entire system, the individual classes became slightly less complex on average.

As a result of the analysis conducted by E2A on the PowerMock system, we identified various periods in the history of the system that were important from the source quality perspective. We also examined also the refactoring mechanisms applied by the developers in response to deteriorating code quality and evaluated their effectiveness.

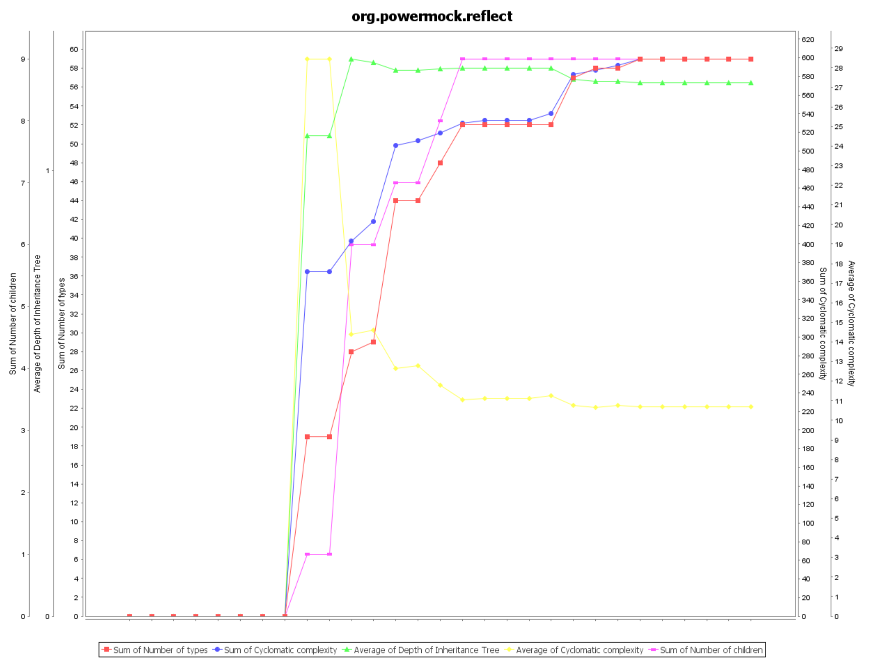


Fig 10.3. Complexity of package reflect, by release.

10.4 Related work

The growing importance of software maintenance stimulated development of several tools for analyzing the evolution of software projects, both in industry and academia. Selected tools are briefly described below.

SonarQube [SQ] is probably the most popular platform of this kind, aiming at monitoring the quality during development. It supports several programming languages and offers various reports on different attributes of code quality. It can be also extended through a designated API and integrated with most popular continuous integration and build tools, which attracted the developers community.

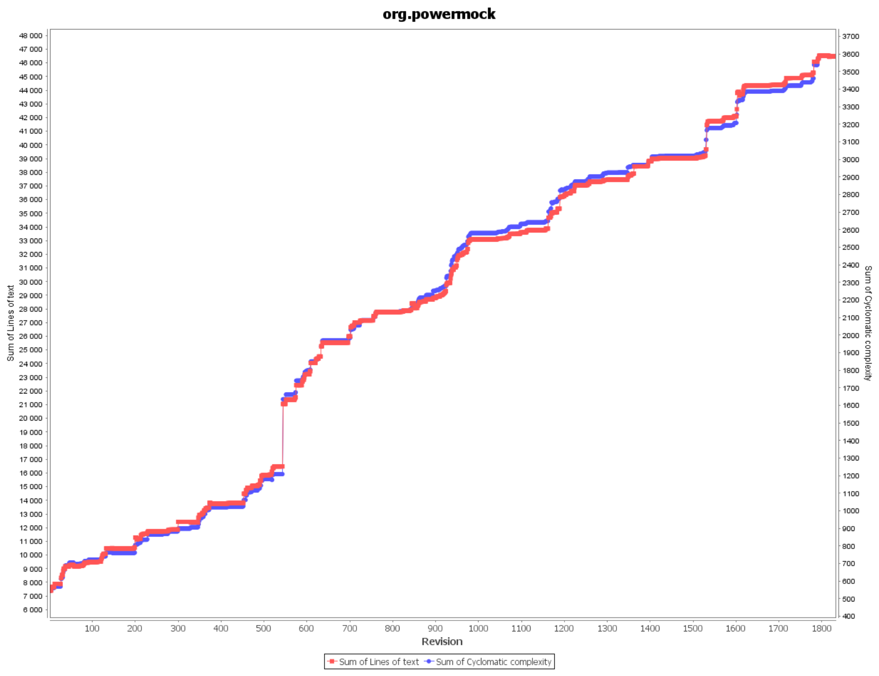


Figure 10.4. Complexity of package modules, by release.

Klocwork Insight [KI] is a commercial tool for monitoring quality of code in popular programming languages. It is focused mainly on discovering defects, analyzing possible security flaws and visualizing the trends.

Several fault-detection and monitoring features are present in several Parasoft tools [PAR]. However, they are focused mostly on the security and error-prevention, not directly on the code quality and maintainability.

Kenyon [BWKG05] is an example of a research tool, aimed at providing a universal platform for collecting and pre-processing data for further analysis in external applications. It is based on slightly different notions when compared to E2A, but the approaches are similar.

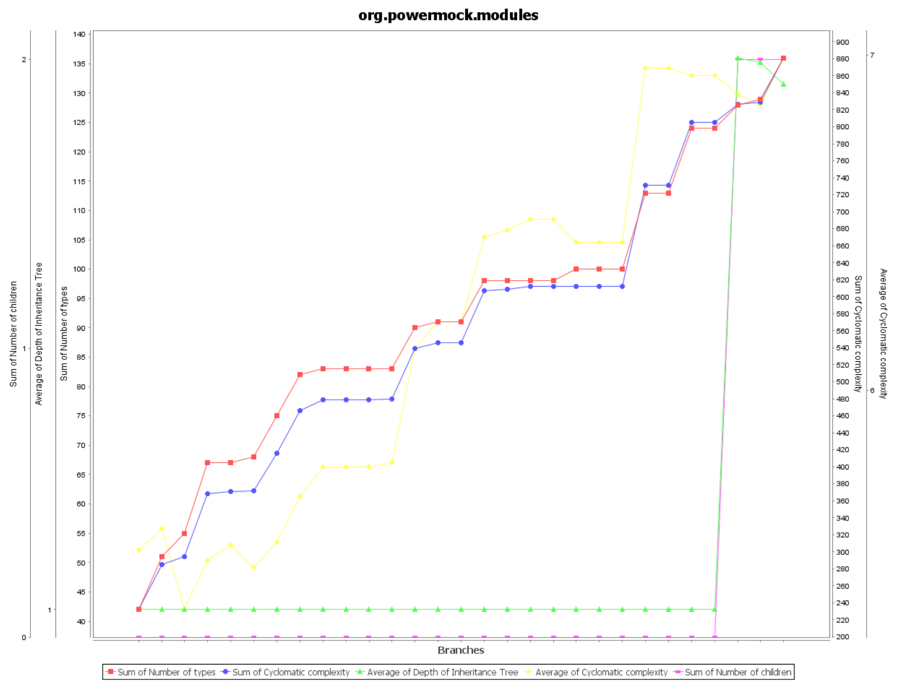


Figure10.5. System size, by revision.

10.5 Summary

In this work we presented E2A – an extensible tool for collecting, analyzing and visualizing the data from SVN repositories. We showed how it can be used in analysis of history of a software project, which helped us to understand and interpret evolution processes that the system underwent. The system also features an extension mechanism that allows for defining and embedding new metrics calculators. This helps researchers in experimenting at the study of software evolution.

References

[Abr95]	F. B. e Abreu: "The MOOD Metrics Set" Proc. ECOOP'95 Workshop on Metrics, 1995.
[Bec99]	K. Beck: “Extreme Programming Explained”. Addison-Wesley, 1999.
[BW84]	V. Basili and D. Weiss. A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering, 10(3):728-738, Nov 1984.
[BWKG05]	J. Bevan, E. J. Whitehead, S. Kim, M. Godfrey: “Facilitating Software Evolution Research with Keynon”. Proc. Of ESEC-FSE, 2005, pp. 177-186.
[CK94]	S. R. Chidamber and C. F. Kemerer: “A metrics suite for object oriented design. IEEE Transactions on Software Engineering”, 20(6):476-493, 1994.
[EMP]	Eclipse Metrics plugin website, http://eclipsemetrics.sf.net (visited May 2014)
[FBBO+99]	M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts: “Refactoring: Improving the Design of Existing Code”. Addison-Wesley, 1999
[GS09]	G. Gousios, D. Spinellis: “Alitheia Core: An Extensible Software Quality Monitoring Platform”. Proc. of 31st ICSE. IEEE, 2009, pp. 579-582.
[KI]	Klocwork Insight website, http://www.klocwork.com/products/insight/ (visited May 2014).
[Leh96]	M. M. Lehman: “Laws of software evolution revisited”. Software Process Technology. LNCS 1149, 1996, pp 108-124.
[Mar04]	R. Marinescu: “Detection Strategies: Metric Based Rules for Detecting Design Flaws”. Proc. of 20th ICSM, 2004, pp. 350-359.
[PAR]	Parasoft website, http://www.parasoft.com/ (visited May 2014).
[PMD]	PMD website, http://pmd.sf.net (visited May 2014).
[SQ]	SonarQube website, http://www.sonarqube.org/ (visited May 2014).

Chapter 11

Can the source code be reviewed on a smartphone?

Although code reviews are a well-known practice with a long tradition, they were not widely used in software industry. One of the reasons for this may be the fact that they are not most enjoyable engineering task when compared to design and coding. Currently, with a number of software tools supporting desktop or web-based reviews which makes the review more comfortable and pleasant task, it seems that the situation improves. The goal of this chapter is to go further and examine the concept of code reviews performed on mobile devices. The chapter describes the developed dedicated Android tool for code reviews, overviews its experimental evaluation and reports some promising results.

One of the key challenges in software development is maintaining source code quality, which comprises several aspects, such as meeting customer requirements, performance, extensibility or readability. For these aspects software industry developed dedicated techniques and tools. For instance functional integrity is well covered by testing and profiling tools and practices which are still developed. On the other hand, supporting code quality in terms of its design and readability is much more difficult aspect. While some tools are available (e.g. checkstyle), still the most appraised practice are regular code reviews (in its various forms) [Coh06], which are known in software industry for nearly four decades.

As every practice, code review has its flaws. Probably the greatest flaw is lack of proper process support and management encouragement. While every developer knows the practice in theory, it is still seldom used in an average software team. Fortunately, current trends in software industry seem favourable for code review by providing tools such as Gerrit [Mil13] or technique of *Pull Requests* which not only technically facilitate the review but also impose it into the development process.

This chapter contributes to these trends with the idea of code review performed on mobile devices, which are ubiquitous these days. The main purpose behind the idea is to encourage developers to bring more focus to code quality and frequent reviews by making them easily available on a train during business trips or in a comfortable sofa in a company social room. Having considered the physical limitations of mobile devices several questions arise on the efficiency of such reviews, including small screens and uncomfortable virtual keyboards. This chapter provides preliminary answers to these questions by presenting early findings on the mobile reviews efficiency and quality based on the experimental comparison of the mobile and classic desktop-based reviews.

11.1. Code review and related work

Code review practice origins from formal inspections proposed by M. Fagan [Fag76]. An inspection is defined as stage-based process which can be applied to all artefacts of software project including documentation, source code or tests. The process is driven by meetings (repeated endlessly until artefacts are considered completed), which are attended by team members assigned to specific roles including authors, reviewers and moderators. Fagan claims that code inspections allow to spare 54 hours per each 1000 lines of code. These claims are confirmed by other, much more contemporary, studies as well [Rad04, Coh06]. Code inspections are also praised [Wel10] for their learning and knowledge sharing effects which greatly speed up development and reduce project risk in case of core developers leave the team.

While code inspections seem valuable, they are also perceived as expensive. [Rad04, FBV05] indicate costs as one of the main reasons for which inspections are not more widely used in software development. To cut the costs, developers experiment with more lightweight variants of Fagan's inspections. This results in different types of inspections leading to inspection types and new terminology [Coh06], in which *inspections* are replaced by *code review* or *peer review*.

Peer reviews do not assume any phases of development. They are performed in *code-assess-respond* style instead. Each piece of work that is done must be reviewed by a peer – strictly speaking, another developer in a team. This approach is more flexible and can be adjusted to internal team process and preferences.

Even though peer reviews are less expensive and more flexible than formal Fagan inspections, they are not widely used neither. [Rad04] suggests that the fundamental reason for this may be the fact that they are not most enjoyable engineering task when compared to design and coding.

Fortunately, over recent years software industry has developed a more positive approach to reviews. To make the developers life easier several dedicated tools are in widely use such as Gerrit [Mil13], Attlasian Crucible, CodeRemarks[1] and others. They facilitate the process by displaying code with an ability to review it by publishing comments. Added remarks are sent to the code's author who can respond to them and fix the mistakes.

[1] *http://www.coderemarks.com/*

Specialized code review application are not the only choice these days. Patches or pull requests from version control system can be subject of reviews too. Version control systems like Git [LM12] or SVN allow to create excerpts with list of changes to be made to the code. Such lists can be reviewed before they are merged to the code, incorporating all code review benefits. The major flaw when using this method is that subsequent pull requests or patches can be hardly compared with each other.

It seems that the proper tooling support improves the *enjoyment factor* of the reviews and developers are more keen to perform it. Thus, this chapter proposes a step further: to support code reviews on mobile devices which got common these days. It is observed that all existing solutions are designed for typical developer environment which is desktop or laptop. On the other hand, code review is more about reading than writing and therefore can have more in common with studying a book than coding. Taking developer out of his usual workplace to social room or comfortable sofa should have a positive impact on his/her attitude to this practice. However, a question arise whether mobile code review will be as efficient as performed on a desktop. This chapter aims at providing preliminary experimental findings in this matter. No other similar or even related research is reported in literature so far.

11.2. Applied research method

To examine the effectiveness of mobile reviews, a comparison of mobile and desktop reviews needs to be performed. That is why the applied research method consists of two following elements:

1. dedicated review tool for mobile device created for the purpose of the research
2. experimental comparison of the results obtained from mobile reviews and desktop reviews

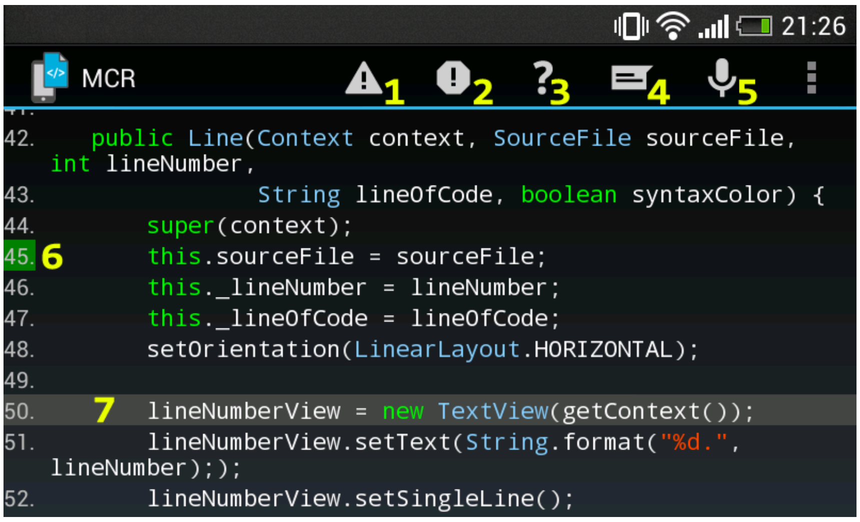
Both elements are described in detail in consecutive subsections.

11.2.1. Tool for mobile code reviews

The developed tool is targeted at Android platform being the most popular in Poland at the moment. This popularity was crucial for the experiment purposes to make sure we will get good balance between mobile and desktop reviews. The screenshot of the working application is presented in Figure 11.1.

The most important functions can summarized as follows:

— displaying source code with syntax coloring and line numbering



```
    public Line(Context context, SourceFile sourceFile,
  int lineNumber,
              String lineOfCode, boolean syntaxColor) {
        super(context);
        this.sourceFile = sourceFile;
        this._lineNumber = lineNumber;
        this._lineOfCode = lineOfCode;
        setOrientation(LinearLayout.HORIZONTAL);

        lineNumberView = new TextView(getContext());
        lineNumberView.setText(String.format("%d.",
  lineNumber););
        lineNumberView.setSingleLine();
```

Figure 11.1. Source code displayed in mobile application

— selecting lines (see item #7 in Figure 11.1) and attaching comment to it (by using icons from #1 to #5 in Figure 11.1)
— marking lines that have comments (green background of line number - see item #6 in Figure 11.1)
— sharing comments with others

To overcome the lack of physical keyboard and improve the comfort of reviewer's work some innovations are introduced to the presented solution. The most important one are predefined reviewer's comments which are easily accessible from top application menu (icons #1 to #3 in Figure 11.1) with one finger touch. In addition, it is possible to record voice comments and hook them to specific line of code (icon #5 in Figure 11.1). These features are created in order to benefit as much as possible from the capabilities of modern mobile devices. They also minimize the inconvenience of typing in the comments when using virtual on-screen keyboard (icon #4 in Figure 11.1).

As for sharing comments, they can be distributed with any Android service that allows to share files - including e-mail, Bluetooth or Wi-Fi.

11.2.2. Experimental comparison

The prepared experiment was designed specifically for comparison of the classic (desktop) and mobile review.

Participants

The participants consisted of Computer Science students from 3rd and 4th year of BSc studies degree from the University of Science and Technology. The study involved 55 programmers, 23 of whom did code review using their own mobile devices, mainly smartphones. This allowed to obtain some diversity in screen resolution and inspect its influence on the work comfort.

Students were educated in code smells and review process. They know Java which mitigates the risk of failing the review task due to a lack of syntax or semantics knowledge of a language. Short introduction to examined review tools has been performed before the experiment.

Compared tools

Each participant performed a code review either on mobile device or on PC computer using a CodeRemarks online tool for code review. The decision whether a particular student should use a mobile device or PC was left to himself, not to impose specific environment in which he could feel uncomfortable. Both mobile phones and tablets were allowed.

CodeRemarks has been chosen for desktop reviews because it does not need any installation or configuration on a reviewer machine. It offers features comparable to the mobile application (display one source file, add text comments). Each reviewer has been given unique URL address where he could review the code. It made both distributing the task and collecting results simple.

Obviously, developed Android application was prepared for the experiment needs. The prepared version could only display source code that was the subject of study. After the timeout had been reached, application automatically sent comments to the author, including screen size and orientation of device when the code review was performed.

Timing

Organization of volunteers for an experiment is always a problem for researches. Therefore, to obtain a reasonable number of participants, it was decided to carry out the review task during student classrooms. This enforced a strict time limit. Based on a few initial (testing) reviews without a timeout it was decided that seven minutes should suffice for the needs of the preliminary evaluation. It is far less time that should be dedicated for review of such code but should be enough for initial comparison of the PC and mobile code review efficiency.

Gathered data

The following data was gather for each participant:

- list of review comments including line number, type and content of the comment.
- screen resolution (for Android reviews only)
- screen orientation (for Android reviews only)
- review duration

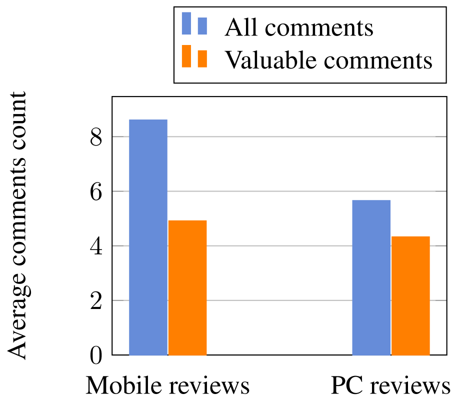


Figure 11.2. Average number of comments

Task

Each participant obtained the same task which was a prepared Java class to be reviewed. The source code of the Java class that was the subject of the study is presented in Appendix 1, Listing 1. The code was a fragment of real program and was deliberately ”enhanced” to contain more defects (code smells). As a result, 120 lines of code contain 30 code smells, 26 of which were known before the experiment (the rest 4 were identified by the participants). The injected code smells were based on a list published in *Clean Code* [Mar09] and on authors' experience.

11.3. Results

The experiment reviews were collected and analyzed after the experiment was done. Each of the reviewer comment was labeled as *valuable* (when it pointed to a recognized code smell) or *worthless* (when the comment could not be understood or did not point to any code smell). Several different aspects were inspected during experiment analysis. The most interesting ones are presented here.

11.3.1. The number of comments

The first aspect is the average total number of comments added in one review session. Average number of comments per review for Android tool is 8 while for desktop it is below 6. However, when only valuable comments are taken into

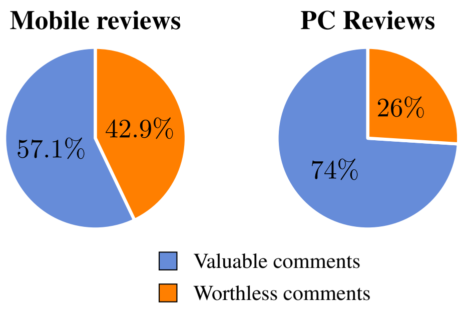


Figure 11.3. Comparison of valuable and worthless comments

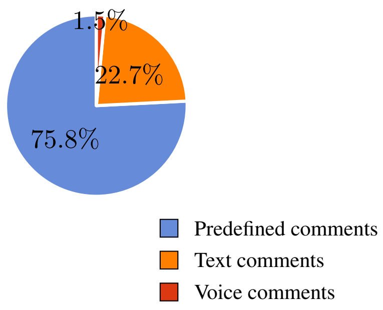


Figure 11.4. Type of comments added when using mobile application

consideration the difference is much smaller and fluctuates around 4.5 comments per review. Quantities are visualized in Figure 11.2.

While the number of detected code smells in one review is comparable for both types of review, it can not be argued that mobile devices introduces unnecessary clutter by many worthless remarks. Relation of valuable and worthless comments added to the code are visualized in Figure 11.3.

It might be expected that usability of mobile application for code reviews is relatively small when compared to large PC's screen.

In order to verify this statement we inspected the number of comments which were misplaced (i.e. assigned to a wrong line number, but the comment's content would still allow the programmer to understand it).

Results do not differ much in both types of reviews. There are 9.3% and 9.9% of misplaced remarks in mobile and PC reviews respectively.

11.3.2. Mobile application usage

The majority of examined Android reviewers performed reviews in vertical screen orientation (75.8%). Devices that were held vertically are tablets with high screen resolution. They still allow to display the whole lines of code without line breaking.

We tried to investigate relation between screen resolution and comments quantity or value, but the results have not reveal any pattern.

As for the type of the comments that were attached to the source code, their frequency is visualized in Figure 11.4. The vast majority of students used predefined comments feature. There were few participants that added voice comment.

11.3.3. Detected code smells

Four of the code smells that had been injected into the reviewed source code have not been found. These smells are:

— long list of imports not shortened with wildcard import (J1[2]) - lines from 21 to 24 on the Listing 1
— overridden safety by unnecessary definition of `serialVersionUID` for `Serializable` class (G4) – line 31
— flag (`boolean`) argument should be replaced by another method (F3, G15) – line 48
— hidden temporal coupling of methods (G31) – lines 61 and 62

However, reviewers managed to find four new flaws that were unknown at the time of preparing the task:

— *"reinventing the wheel"* by checking `list.size() > 0` instead of using `list.isEmpty()` from `java.util.List` interface – lines 80 and 94
— inconsistent code formatting in two places (G24) – lines from 104 to 107 and line 116
— no defensive copy of collection returned by a method - line 113

The most conspicuous flaws had been found the most frequently. Three of the most recognizable code smells are shown in Table 11.1.

Another examined aspect is the first code smell that was recognized in every review session. Surprisingly, there are only 7 *first-smells* on mobile code reviews whereas on PCs there are 12.

[2] codes of code smells as defined in *Clean Code* [Mar09]

Table 11.1. Code defects with the highest detection rate.

Defect	Lines	Detection (mobile)	Detection (PC)
unnecessary underscore prefixes for private class fields (N6)	34, 36	61%	47%
commented out, obsolete code (C5)	99 - 101	52%	31%
ambiguous method name (N1, N4)	67	43%	66%

11.4. Discussion

The summarized results presented in Section 11.3 allow to make several interesting observations.

11.4.1. Mobile code reviews might need more time to become popular

Statistics of comments correctness shown in Figure 11.3 indicates, that there were far too many worthless comments in mobile code reviews. There can be a few reasons of such situation:

— participants have seen the mobile application for the first time, so they might have wanted to test how it works by adding random comment,
— they were unwilling to perform code review on small screens,
— students, of whom the majority of participants consisted, might have not paid much attention to quality of the review.

While web- or desktop-based reviews are proven to be effective, mobile code reviews have not been tried before. Thus, in some cases developers require time to make their minds and use the mobile application. This is the argument that appeared the most often when asking participants whether they would use such application in their work.

11.4.2. Code reviews are effective regardless of the method of their performing

Having ignored worthless comments, both PC and mobile reviews are comparable when the quantity of found code smells in one session is considered. It means that mobile reviews are not worse than PC reviews.

As for the efficiency of code reviews, the average number of code smells found in both types of reviews is equal to 4.68. Comparing it to the total number of known defects of the reviewed code we get the amount of 15.6% code smells found in one review. It is much fewer than the expected value of 60% of defects being detected when practicing code reviews [MVM09].

However, when the time limit is taken into consideration, it is clear that reviewers were not able to find expected amount of code smells. The most efficient reviews need to last 88 minutes per 188 lines of code [MVM09]. Therefore, reviewer would need 56 minutes for 120 lines of code. Experiment gave only seven minutes, so we can calculate expected amount of defects found in conducted study as:

$$x = \frac{7}{56} \approx 13\% \qquad (1)$$

The value is much more appropriate now and proves both code reviews efficiency and the described ideal speed of performing them.

11.4.3. Mobile code reviews are not uncomfortable

This statement is formed mainly based on almost the same frequency of misplaced comments in PC and mobile reviews. The main reason for gathering such information about remarks added to the code was belief that results of mobile reviews would contain many comments added to a wrong line. However, performed analysis did not prove this expectation. It turns out that it is equally easy to tap the wrong line on touch display and to click it on PC's screen. Given also predefined and voice comments on mobile application, it can be even more comfortable tool to use.

11.4.4. Predefined comments are promising

Predefined comments were the most frequently added when using mobile application. It is surprising that such feature is not common in existing tools supporting code reviews. Participants were excited about possibility of adding a comment to the code by single tap. In this way, they can make the review much quicker and in more comfortable manner.

There were cases when predefined comment added to the code did not address code smell exactly. As an example, predefined comment *"Poor naming"* added to the line with useless `TWO = 2` constant does not express clearly the intention of the reviewer on how improve the code. However, there is a high probability that the author of the code reading such remark would think about this *poor naming* comment which should lead to removal (or renaming) it. Therefore, such comments in the study have been marked as valuable.

During the experiment, application had the predefined comments hardcoded. Many participants suggested allowing to add own predefined comments on project basis. Such enhancement would allow to adjust vocabulary and common types of defects to the reviewer which would make them even more useful.

11.4.5. Voice comments introduce communication problems

Very few participants used feature of voice comments. It indicates some problems with this technique.

Firstly, the way of performing the study should be taken into account. Most of the reviews were done simultaneously in a group of students. Recording a voice comment in public can be perceived as weird behavior when everybody hears what reviewer thinks about particular fragment of a code. Moreover, if each participant would start to record a voice comment, they would be hard to understand.

It seems that this can affect also real developers who work in co-located teams. However, for freelance, open-source or distributed developers this way of performing reviews can provide some practical value.

11.4.6. Small screen enhances details

Interestingly, there are some defects that were found during mobile reviews only. These are:

- typographic error (`recodedFile` as method argument name instead of `recordedFile`) – line 91 on Listing 1
- inconsistent formatting of code – lack of space before opening curly bracket – line 116

This may lead to a conclusion that displaying source code on small screens enforces reviewers to pay more attention on details of the code. When seeing only a few lines of code at the moment, they can focus more on such trivialities.

On the other hand, small screen prevents reviewers from seeing the whole picture of a code being reviewed. Semantic error in the code has been found only during PC reviews. It required detail analysis of the entire class and therefore it was hard to be detected on mobile devices.

11.4.7. Small screen encourages reviewers to analyse code line by line

The first code smells that were detected reveal a pattern which was used to perform code reviews on both devices. Defects placed in the beginning of the file (lines from 1 to 48) were found as first-smells in over 70% of mobile reviews and in almost 50% of PC reviews. It indicates that reviewing with mobile application enforces to analyse source code line by line.

11.5. Conclusions and further work

The aim of this research was to inspect the aspect of code reviews in the context of mobile devices. There are two important conclusions that need to be drawn from this research.

First of all, mobile reviews are proven to be as efficient as classic (desktop) code reviews we do these days. Developed prototype Android tool for code reviews and organized experiment prove that in some aspects mobile reviews can be even more convenient than desktop reviews. Having agreed that the comfort and *enjoyment factor* are important for developers [Rad04], we believe that mobile reviews will find their practical application in software industry in near future.

Secondly, the conducted study confirmed that code reviews are really an effective way of ensuring quality of a source code. Almost all of the code smells that were prepared for the experiment were found and the collected comments were assessed as correct and helpful in fixing the errors and making the code more readable and clean.

Further research should include organization of longer experiment with more participants involved in order to confirm the preliminary results and observations. An interesting aspect would be also to compare both types of reviews in the context of a complex code structure. In this way, the observation that the small screen of mobile device limits the reviewer in obtaining a bigger picture of the source code could be verified.

What is more, it is planned to deploy the prototype Android tool in a real development environment. A good candidate is a project developed at University of Science and Technology for Government Protection Bureau (funded by Polish National Centre for Research and Development). The development team utilizes Gerrit code review application and it already tried out the prototype Android tool with positive feedback. The deployment will allow for comparison of both tools (Gerrit vs Android tool) which can produce interesting results.

For this purpose the Android tool must be further enhanced. The most demanded features are:

— ability to open many files at once
— *diff* feature allowing to display previous and current version of the code
— support for version control systems
— support for communication with existing code review tools for desktop

Acknowledgements

The research leading to these results has received funding from the research project No. DOBR-BIO4/060/13423/2013, funded by the Polish National Centre for Research and Development.

References

[Coh06] J.A. Cohen. *Best Kept Secrets of Peer Code Review*. Printing Systems, 2006.

[Fag76] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182–211, 1976.

[FBV05] Bernd Freimut, Lionel C. Briand, and Ferdinand Vollei. Determining inspection cost-effectiveness by combining project data and expert opinion. *IEEE Transactions on Software Engineering*, 31(12):1074–1092, 2005.

[LM12] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, 2012.

[Mar09] Robert C. Martin. *Clean Code, A Handbook of Agile Software Craftsmanship*. Pearson Education, Inc., 2009.

[Mil13] Luca Milanesio. *Learning Gerrit Code Review*. Packt Publishing, 2013.

[MVM09] Casper Lassenius Mika V. Mäntylä. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, 35(3):5, 2009.

[Rad04] R.A. Radice. *High Quality Low Cost Software Inspections*. Paradoxicon Publishing, 2004.

[Wel10] Lisa Wells. 9 reasons to review code. *http://blog.smartbear.com/software-quality/9-reasons-to-review-code/*, 2010.

Appendix 1 - Source code reviewed during experiment

```
package pl.fracz.mcr.source;

/*
 2013-10-23, fracz, first implementation
 2013-10-30, fracz, added syntax highlighting
 2014-02-26, fracz, added ability to add voice comment
 */

import android.annotation.SuppressLint;
import android.content.Context;
import android.graphics.Color;
import android.graphics.Typeface;
import android.text.Html;
import android.widget.LinearLayout;
import android.widget.TextView;

import java.io.File;
import java.io.Serializable;
import java.util.List;

import pl.fracz.mcr.comment.Comment;
import pl.fracz.mcr.comment.CommentNotAddedException;
import pl.fracz.mcr.comment.TextComment;
import pl.fracz.mcr.comment.VoiceComment;

/**
 * View that represents one line of code.
 */
@SuppressLint("ViewConstructor")
public class Line extends LinearLayout implements Serializable {
   private static final long serialVersionUID = 3076583280108678995L;
   private static final int TWO = 2;

   private final int _lineNumber;

   private final String _lineOfCode;

   // holds the line number
   private final TextView lineNumberView;

   private final TextView lineContent;

   private final SourceFile sourceFile;

   private List<Comment> comments;

   public Line(Context context, SourceFile sourceFile, int lineNumber,
            String lineOfCode, boolean syntaxColor) {
      super(context);
      this.sourceFile = sourceFile;
```

```
        this._lineNumber = lineNumber;
        this._lineOfCode = lineOfCode;
        setOrientation(LinearLayout.HORIZONTAL);

        lineNumberView = new TextView(getContext());
        lineNumberView.setText(String.format("%d.", lineNumber););
        lineNumberView.setSingleLine();
        lineNumberView.setWidth(30);
        addView(lineNumberView);

        TextView lineContent = new TextView(getContext());
        addLineContent(syntaxColor);

        this.comments = sourceFile.getComments().getComments(this);
    }

    public int get() {
        return _lineNumber;
    }

    /**
     * Adds a text comment.
     *
     * @param comment
     * @throws CommentNotAddedException
     */
    public void addTextComment(String comment) throws
        ↪ CommentNotAddedException {
        sourceFile.getComments().addComment(this, new TextComment(comment));
        this.comments = sourceFile.getComments().getComments(this);
        if (comments.size() > 0) {
            lineNumberView.setBackgroundColor(Color.parseColor("#008000"));
        }
    }

    /**
     * Adds a voice comment.
     *
     * @param recordedFile
     * @throws CommentNotAddedException
     */
    public void createVoiceComment(File recodedFile) throws
        ↪ CommentNotAddedException {
        sourceFile.getComments().addComment(this, new
            ↪ VoiceComment(recodedFile));
        this.comments = sourceFile.getComments().getComments(this);
        if (comments.size() > 0) {
            lineNumberView.setBackgroundColor(Color.parseColor("#008000"));
        }
    }

//   public void addVideoComment(File videoFile) throws
    ↪ CommentNotAddedException {
```

```
//
//   }

   private void addLineContent(boolean syntaxColor){
      if (!syntaxColor ||
          ↪ !SyntaxHighlighter.canBeHighlighted(syntaxColor))
         lineContent.setText(Html.fromHtml(lineOfCode));
      else
         lineContent.setText(SyntaxHighlighter.highlight(Html.
             ↪ fromHtml(lineOfCode)));
      lineContent.setTypeface(Typeface.MONOSPACE);
      addView(lineContent);
   }

   public List<Comment> getComments() {
      return this.comments;
   }

   public boolean hasConversation(){
      sourceFile.markConversation(this);
      return getComments().size() > TWO;
   }
}
```

Listing 1. Source code reviewed during experiment

Authors and affiliations

Preface
Lech Madeyski
Wroclaw University of Technology, Faculty of Computer Science and Management, Institute of Informatics
lech.madeyski@pwr.edu.pl
Mirosław Ochodek
Poznan University of Technology, Faculty of Computing
miroslaw.ochodek@cs.put.poznan.pl

CHAPTER 1
Jakub Jurkiewicz
Poznan University of Technology, Faculty of Computing
jakub.jurkiewicz@cs.put.poznan.pl
Piotr Kosiuczenko
Military University of Technology in Warsaw, Department of Cybernetics
pkosoiczenko@wat.edu.pl
Lech Madeyski
Wroclaw University of Technology, Faculty of Computer Science and Management, Institute of Informatics
lech.madeyski@pwr.edu.pl
Mirosław Ochodek
Poznan University of Technology, Faculty of Computing
miroslaw.ochodek@cs.put.poznan.pl
Cezary Orłowski
IBM Center for Advanced Studies on Campus, Gdańsk,
cor@zie.pg.gda.pl
Łukasz Radliński
West Pomeranian University of Technology,
Faculty of Computer Science and Information Technology
lukasz.radlinski@zut.edu.pl

CHAPTER 2
Marek Majchrzak
Capgemini Poland
Wroclaw University of Technology, Faculty of Computer Science and Management, Institute of Informatics
marek.majchrzak@capgemini.com

Łukasz Stilger
Capgemini Poland
lukasz.stilger@capgemini.com
Marek Matczak
Capgemini Poland
marek.matczak@capgemini.com

CHAPTER 3

Tomasz Sitek
Gdańsk University of Technology, Faculty of Management and Economics
tsitek@zie.pg.gda.pl
Artur Ziółkowski
Gdańsk University of Technology, Faculty of Management and Economics
aziolko@zie.pg.gda.pl

CHAPTER 4

Miklós Biró
Software Competence Center Hagenberg,
miklos.biro@scch.at

CHAPTER 5

Stanisław Jerzy Niepostyn
Warsaw University of Technology, Institute of Computer Science
S.Niepostyn@ii.pw.edu.pl
Andrzej Tyrowicz
Agencja Europejska, Andrzej Tyrowicz
a@tyrowicz.eu

CHAPTER 6

Łukasz Radliński
West Pomeranian University of Technology,
Faculty of Computer Science and Information Technology
lukasz.radlinski@zut.edu.pl

CHAPTER 7
Bogumiła Hnatkowska
Wroclaw University of Technology, Faculty of Computer Science and Management, Institute of Informatics
Bogumila.Hnatkowska@pwr.edu.pl
Łukasz Wrona
Wroclaw University of Technology, Faculty of Computer Science and Management, Institute of Informatics
lukwro83@gmail.com

CHAPTER 8
Andrzej Ratkowski
Warsaw University of Technology,
Institute of Control and Computation Engineering
Krzysztof Gawryś
Warsaw University of Technology,
Institute of Control and Computation Engineering
Eliza Świątek
Warsaw University of Technology,
Institute of Control and Computation Engineering

CHAPTER 9
Ilona Bluemke
Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Computer Science
I.Bluemke@ii.pw.edu.pl
Anna Stepień
Warsaw University of Technology, Faculty of Electronics and Information Technology, Institute of Computer Science
A.Stepien.1@stud.elka.pw.edu.pl

CHAPTER 10
Michał Ćmil
Poznan University of Technology, Faculty of Computing
cmilmichal@gmail.com
Bartosz Walter
Poznan University of Technology, Faculty of Computing
bartosz.walter@cs.put.poznan.pl

CHAPTER 11

Wojciech Frącz
AGH University of Science and Technology
fracz@iisg.agh.edu.pl
Jacek Dajda
AGH University of Science and Technology
dajda@agh.edu.pl