

Matthias Naab

Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems



Editor-in-Chief: Prof. Dr. Dieter Rombach
Editorial Board: Prof. Dr. Frank Bomarius
Prof. Dr. Peter Liggesmeyer
Prof. Dr. Dieter Rombach

FRAUNHOFER VERLAG

PhD Theses in Experimental Software Engineering

Volume 41

Editor-in-Chief: Prof. Dr. Dieter Rombach

Editorial Board: Prof. Dr. Frank Bomarius
Prof. Dr. Peter Liggesmeyer
Prof. Dr. Dieter Rombach

Contact:

Fraunhofer-Institut für Experimentelles Software Engineering (IESE)
Fraunhofer-Platz 1
67663 Kaiserslautern
Telefon +49 631 6800 - 0
Fax +49 631 6800 - 1199
E-Mail info@iese.fraunhofer.de
www.iese.fraunhofer.de

Bibliographic information published by Die Deutsche Bibliothek

Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.d-nb.de>.
ISBN: 978-3-8396-0477-9

D 386

Zugl.: Kaiserslautern, Univ., Diss., 2012

Printing and Bindery:
Mediendienstleistungen des
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Printed on acid-free and chlorine-free bleached paper.

© by **FRAUNHOFER VERLAG**, 2012

Fraunhofer Information-Centre for Regional Planning and Building Construction IRB
P.O. Box 80 04 69, D-70504 Stuttgart
Nobelstrasse 12, D-70569 Stuttgart
Phone +49 (0) 711 970-2500
Fax +49 (0) 711 970-2508
E-Mail verlag@fraunhofer.de
URL <http://verlag.fraunhofer.de>

All rights reserved; no part of this publication may be translated, reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. The quotation of those designations in whatever way does not imply the conclusion that the use of those designations is legal without the consent of the owner of the trademark.

Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems

Beim Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation
von

Dipl.-Inf. Matthias Naab

Fraunhofer Institut für Experimentelles Software Engineering (IESE)
Technische Universität Kaiserslautern

Berichterstatter:	Prof. Dr. Dr. h.c. Dieter Rombach Prof. Dr. Ralf Reussner
Dekan:	Prof. Dr. Arnd Poetzsch-Heffter
Tag der Wissenschaftlichen Aussprache:	02.10.2012

D 386

Acknowledgement

I would like to express my gratitude to many people who supported me in the last years while doing my PhD.

I thank my supervisors Prof. Dieter Rombach and Prof. Ralf Reussner for their advice and support. Working at Fraunhofer IESE gave me many opportunities and helped me to learn much about software engineering. I would like to thank Dirk Muthig for his support in shaping my PhD topic in the early days of the thesis. Further, I would like to thank Jörg Dörr, Marcus Trapp, and Thorsten Keuler for their feedback on the thesis.

At Fraunhofer IESE, I worked with many outstanding colleagues and we had great discussions, on my PhD topic, on software architecture as well as on other topics. I would like to thank all colleagues of the PLA, IS, and ISD departments. In particular, I would like to express my thanks to Sebastian Adam, Michalis Anastasopoulos, Ralf Carbon, Jörg Dörr, Thomas Forster, Thorsten Keuler, Jens Knodel, Dirk Muthig, Marcus Trapp, and the whole software architecture team. When getting started with my thesis, we had fruitful internal PhD meetings with the colleagues of the PLA department, in which I particularly benefited from the feedback of Michalis Anastasopoulos, Thorsten Keuler, and Jens Knodel.

I would like to thank all people involved in the validation of my approach. Anne Groß helped me to organize the experiment, in which 17 students from a practical course at the Technical University of Kaiserslautern voluntarily participated. Further, I would like to thank IESE customers who gave me the opportunity to apply and improve my approach in their contexts. Further, I want to thank Christian Webel for the opportunity to align this PhD with ongoing research projects, in particular the ADiWa project.

With Karlsruhe Institute for Technology and FZI, we established a fruitful, regular discussion on software architecture and maintainability. I would like to express my thanks specifically to Prof. Ralf Reussner and Johannes Stammel for the intense discussions on PhD-related topics.

Last, but far from least, I would like to thank my family and friends, who had to spend many evenings and weekends without me. I would like to thank my wife Esther and my parents Gertrud and Hans-Peter for supporting me in any possible way. I thank my sister Judith for proof-reading this thesis. Finally, I thank my daughter Anna-Lena who strongly motivated me to bring this thesis to an end before she was born. In the end she was born just when I was writing the very final pages of my thesis.

Abstract

Flexibility is an indispensable quality attribute of long-living information systems. Today's enterprises heavily rely on information systems for running their businesses. In domains like banking, insurance, or aviation, information systems are even a core enabler of competitiveness. In a dynamic business world, requirements evolve and software has to follow. How much implementation effort a change requires is strongly impacted by a system's architecture. Despite the availability of paradigms like SOA, BPM, or EDA, which come with flexibility mechanisms and are widely expected to bring inherent flexibility, today's systems are often not as flexible as expected. A major reason for missing flexibility is the lack of systematic, constructive support for flexibility in architecture definition methods.

An in-depth characterization of the quality attribute flexibility is our foundation for systematically defining flexible architectures for software systems. Particular focus is on the role of architecture and on how it can contribute to a system's flexibility. We introduce a metric for flexibility, measuring on flexibility scenarios and architecture models. We condense key facets of flexibility in a conceptual model.

The key methodical contribution of this thesis is the constructive support for defining flexible architectures. We build on existing architecture definition methods and enhance them. The detailed characterization of flexibility is crucial for providing constructive guidelines and heuristics for architects. Beyond the localization of change impact, the alignment of flexibility mechanisms and business logic is of particular importance for flexibility. Consequently, we support it with design heuristics. Furthermore, we support architects with automated, near-real-time feedback on the achieved level of flexibility, allowing quick corrections of architectural decisions. This is facilitated by a new architectural view, the change impact view, which is modeled by the architect and supports reasoning about flexibility. For paradigms like SOA, we show how they can be leveraged in architecture design to consequently exploit their flexibility potential. This methodical contribution is a conceptual plugin for architecture definition methods which adds specific support for flexibility.

With an implementation of the automated flexibility measurement as an AddIn of Enterprise Architect, we demonstrate the feasibility of this methodical part. Within a controlled experiment we confirmed the hypothesis that architects come up with significantly more flexible architecture designs when they explicitly model change impact views. In projects with industrial customers we experienced the effectiveness, efficiency, and applicability of the contributions and collected qualitative results.

Table of Contents

Acknowledgement	iii
Abstract	v
List of Figures	xi
List of Tables	xiii
1 Introduction	1
1.1 Business Drives IT, and Business Drives (too) Fast	1
1.2 Research Method	5
1.3 Problem Statement	6
1.4 Solution Ideas and Hypotheses	15
1.5 Scope, Context, and Assumptions	21
1.6 Contributions Overview	24
1.7 Thesis Outline	25
2 Foundations of Architecture	27
2.1 Architecting as an Engineering Activity	27
2.1.1 Definitions and Essence	27
2.1.2 Architecting in Practice	28
2.1.3 The ACES Approach	30
2.2 Service-Oriented Architecture	36
2.2.1 Definitions and Essence	36
2.2.2 SOA in Practice	38
2.2.3 A Conceptual Model for Service-Oriented Engineering	41
2.2.4 SOA as Architectural Style and Reference Architecture	44
2.2.5 New Paradigms in Service-Orientation	48
3 Flexibility: State of the Art	49
3.1 Flexibility as a Quality Attribute	50
3.1.1 Flexibility and Related Quality Attributes	50
3.1.2 Flexibility in Information Systems Research	52
3.1.3 Flexibility in other Disciplines	53
3.1.4 Variability in Product Line Engineering	55
3.2 Construction for Flexibility	55
3.2.1 Elicitation of Flexibility Requirements	55
3.2.2 Architecture Definition Approaches	56
3.2.3 Architecture Mechanisms for Flexibility	57
3.2.4 Design Approaches for SOA	58
3.3 Measurement and Evaluation of Flexibility	60
3.3.1 General Overview on Architecture Evaluation Methods	60
3.3.2 Evolution Complexity	61
3.3.3 Analyzing Modifiability at Architecture Level	61
3.3.4 Modifiability and Real Options Theory	62

3.3.5	Palladio and Maintainability Prediction	63
3.3.6	Enterprise Systems Modifiability Analysis.....	63
3.3.7	Further Related Research.....	65
3.4	Flexibility in SOA-Based Information Systems	65
3.5	Summary and Conclusion.....	66
4	Flexibility as a Quality Attribute of Software.....	69
4.1	Characterization of Flexibility.....	70
4.1.1	Principle Characteristics.....	70
4.1.2	Flexibility in the Software System Lifecycle	72
4.1.3	Flexibility and Software Engineering Artifacts.....	73
4.1.4	Flexibility in a Spectrum of Uncertainty.....	74
4.2	Flexibility Requirements	75
4.2.1	Capturing Flexibility Requirements with Scenarios	76
4.2.2	Characterizing and Classifying Flexibility Scenarios	77
4.2.3	Flexibility and Competing Requirements.....	81
4.3	The Role of Architecture for Flexibility	82
4.3.1	Which Architecture Makes a System Flexible?	83
4.3.2	How Does an Architect Make a System Flexible?.....	88
4.3.3	Architecture Meta-Model and Metrics for Flexibility	88
4.3.4	Cost Considerations of Flexibility.....	97
4.4	Conceptual Model of Flexibility	97
5	Engineering Flexible Software Systems	104
5.1	Methodical Overview	104
5.2	Eliciting Flexibility Scenarios.....	107
5.3	Architecture Design for Flexibility	109
5.3.1	Design Goals.....	109
5.3.2	Design Process Overview.....	110
5.3.3	Design Process Activities	113
5.4	Measuring Flexibility with Tool Support	121
5.4.1	Continuously Measuring Flexibility in Architecting.....	121
5.4.2	Features and Exemplary Application of the Tool	123
5.4.3	Realization of the Flexibility-Tool	128
5.5	Discussion	130
6	Flexibility in SOA-Based Information Systems.....	134
6.1	Challenges around Flexibility in SOA.....	135
6.1.1	Typical Flexibility Requirements	136
6.1.2	Characteristics Challenging Flexibility	137
6.2	Architectural Solutions for Flexibility in SOA	138
6.2.1	Architectural Principles in SOA Supporting Flexibility ...	139
6.2.2	Architectural Mechanisms in SOA Supporting Flexibility	139
6.2.3	Key Architectural Considerations for Flexibility in SOA	146
6.3	Technologies Supporting Flexibility in SOA	146

7	Validation.....	148
7.1	Objectives and Hypotheses.....	148
7.2	Controlled Experiment	150
7.2.1	Context of the Experiment.....	150
7.2.2	Setup of the Experiment	151
7.2.3	Analysis and Results.....	156
7.2.4	Observations and Discussion	161
7.2.5	Threats to Validity	163
7.3	Project Experiences.....	166
7.3.1	Project A.....	168
7.3.2	Project B	169
7.3.3	Project C.....	170
8	Summary and Outlook.....	171
8.1	Results and Contributions	171
8.2	Limitations and Future Work	174
8.3	Concluding Remarks	177
	References.....	179
Appendix A	List of Abbreviations.....	191
Appendix B	Experiment Material.....	193
Appendix C	Experiment Raw Data.....	206

List of Figures

Figure 1:	Relationships between business and IT	2
Figure 2:	Research method of the thesis	5
Figure 3:	Cost in IT and business caused by missing flexibility	7
Figure 4:	Set representation of requirements around flexibility	11
Figure 5:	a) CheckIn process b) Simplified architecture of airline system	13
Figure 6:	Illustration of true flexibility	15
Figure 7:	Derivation of industry goals	16
Figure 8:	Research ideas in the context of architecture design	20
Figure 9:	Relationship between research directions, challenges, and ideas	20
Figure 10:	Scope and context of the thesis	21
Figure 11:	Contributions of the thesis in categories	25
Figure 12:	Architecture engagement purposes	31
Figure 13:	Development phases according to RUP [Kru03]	33
Figure 14:	Competence packaging in ACES	34
Figure 15:	Architecture core and domain competence	35
Figure 16:	Key areas of service-orientation	43
Figure 17:	Example view from conceptual model for service-orientation	44
Figure 18:	a) SOA triangle [Erl06] b) SOA element types [KBS04]	45
Figure 19:	a) Solution stack view b) Middleware view [AZE+07a, AZE+07b]	46
Figure 20:	Facets of SOA and their relationships	47
Figure 21:	State-of-the-art in the context of research directions	49
Figure 22:	Relationships among quality attributes	51
Figure 23:	Modifiability meta-model [LFJ+09]	64
Figure 24:	Overview on research ideas of the thesis	68
Figure 25:	Distribution of change effort to change requirements	71
Figure 26:	System lifecycle phases and activities related to flexibility	72
Figure 27:	Flexibility in a spectrum of uncertainty	74
Figure 28:	Characterization of architecture scenarios	77
Figure 29:	Distinguishing the levels business-logic-agnostic and business-logic specific	87
Figure 30:	Architecture meta-model for modules	89
Figure 31:	Principle of measuring flexibility	92
Figure 32:	Flexibility metric function definition	93
Figure 33:	Meta-model for change impact	96
Figure 34:	Views of the conceptual model for flexibility	98
Figure 35:	Conceptual model: flexibility core view	99
Figure 36:	Conceptual model: architecture construction view	100

Figure 37: Conceptual model: architecture implementation view	101
Figure 38: Conceptual model: flexibility measurement view	102
Figure 39: Architecting as activity between requirements engineering and development	105
Figure 40: Contributions to the architecting activities	106
Figure 41: Architecting design process overview	112
Figure 42: Architecting design process overview – key integrations	120
Figure 43: Key contributions to flexibility measurement	122
Figure 44: a) Modeling structural views in EA b) Modeling change impact in EA	126
Figure 45: Flexibility evaluation results	127
Figure 46: a) Flexibility tool configuration b) Matrix showing impacts-relationships	128
Figure 47: Architecture diagram for flexibility AddIn	129
Figure 48: SOA-specific contributions around flexibility	135
Figure 49: SOA architecture mechanisms mapped to SOA technologies	147
Figure 50: Experimental design	153
Figure 51: Measuring flexibility in the experimental results	156

List of Tables

Table 1:	SOA Check 2010: “Which strategic goals does your company aim at with SOA?” [MER10]	4
Table 2:	Scenario characteristics and questions	78
Table 3:	Architecture principles supporting flexibility	84
Table 4:	Architecture mechanisms supporting flexibility	85
Table 5:	Requirements for flexibility metric	91
Table 6:	Architecture example flexibility metrics - element sizes	125
Table 7:	SOA architectural mechanism: Service concept	140
Table 8:	SOA architectural mechanism: Basic service communication	141
Table 9:	SOA architectural mechanism: Service typing	142
Table 10:	SOA architectural mechanism: Separation of services, process logic, UIs	143
Table 11:	SOA architectural mechanism: Descriptive process logic	144
Table 12:	SOA architectural mechanism: Enterprise Service Bus	145
Table 13:	Hypotheses for the areas of contributions	149
Table 14:	Number of valid results per scenario and group	158
Table 15:	Flexibility values achieved (valid ones only) per group	158
Table 16:	Debriefing questionnaire: Results on task-related questions	160
Table 17:	Debriefing questionnaire: Results on flexibility-related questions	161
Table 18:	Experiment raw data: Group A – Briefing Questionnaire	207
Table 19:	Experiment raw data: Group B – Briefing Questionnaire	207
Table 20:	Experiment raw data: Group A – Debriefing Questionnaire	208
Table 21:	Experiment raw data: Group B – Debriefing Questionnaire	208
Table 22:	Experiment raw data: Group A – Flexibility Results	209
Table 23:	Experiment raw data: Group B – Flexibility Results	209

1 Introduction

"It is change, continuing change, inevitable change that is the dominant factor in society today. No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be."
Isaac Asimov

1.1 Business Drives IT, and Business Drives (too) Fast

Business relies on IT	Today, nearly all enterprise organizations heavily rely on IT-systems ¹ to support their businesses in various ways. Information systems handle the increasing amount of data, provide automation of recurring and computation-intensive tasks, support various types of business processes involving single persons or even multiple organizations, and guide people through their IT-supported tasks.
Business drives IT	IT has become an indispensable and costly asset of organizations. The result of this is that IT has conquered a prominent position in enterprises, which leads to the danger of IT becoming an end-in-itself and produces fancy but useless solutions. Thus, it is important that the roles of business and IT are clearly stated and accepted. "Business drives IT" [Len11, AH06] summarizes the relationship that IT always has the responsibility to provide the best possible support for business, and it is widely accepted by practitioners [Wit07, MER10, Bal09]. Of course, there is also a critical relationship in the other direction. "IT drives Business" [Len11] expresses that many of today's business models and business capabilities would not exist without IT and IT is a strongly evolving enabler of business [MER10], forcing business to change for keeping pace with competitors.
Business changes require IT changes	Business is not static. It must continuously change according to internal or external forces in order to stay competitive. That might be to evolve business models and differentiators, standardize parts of the business that become commodity, conduct mergers with other companies, or follow regulatory requirements [Spr05]. "Business drives IT" leads to the

¹ IT-System: Also named "Information System" (IT = Information Technology)

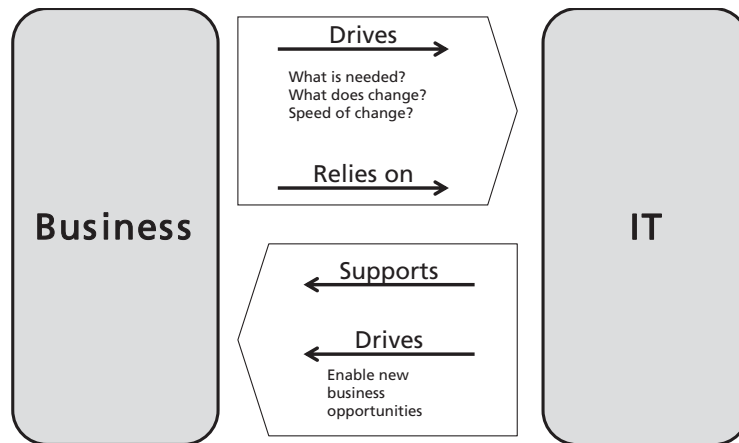


Figure 1: Relationships between business and IT

demand for IT-systems to allow fast and cheap changes. This ability is in particular decisive for IT-systems supporting the competition and differentiation [GBD08] of an enterprise. Today's IT-systems are often old and have accompanied the history of a company and its business. Over time, more and more automation and coverage of business have been achieved and the degree of integration among systems has increased. These changes are often expensive [Par94] and the resulting systems are even more expensive to change.

Flexibility enables IT changes

Flexibility is the property of IT-systems, which expresses how well an IT-system supports certain changes to it that are necessary to follow changes in business. Intuitively, flexibility measures how easy or cheap it is to conduct these changes. In practice, flexibility is widely perceived as a key property of IT-systems [GS06, MER10]. Beyond the pure properties of the involved IT-Systems, an enterprise also needs the ability to conduct the changes of business, organization, and IT in an aligned, controlled, and efficient way. This ability is called "Organizational / Business Agility" [Sch04]. Figure 1 illustrates the relationships between business and IT as described. Beyond the flexibility of a system, there are many other factors that influence how easily changes can be conducted [BAA10].

Example: Airline industry

Airline industry is an apparent example illustrating the aspects described above. The key business of airlines is transporting people or goods from one point to another. To provide this service, a large number of business processes and IT-systems are necessary (excluding the systems in airplanes). Important business processes are booking, check-in, or baggage handling [LSY11a, Ama11]. While in the early days many of these business processes have been done manually, nowadays there is a high and even increasing degree of automation. This can be directly observed when buying tickets on the internet, checking in online or at the desk, and getting baggage delivered quickly and reliable even at huge airports. Certainly, not all airlines follow the same business models. On one end of the spectrum there are airlines offering high-quality service, on the

other end there are airlines offering extremely low prices. However, all different types of business models highly rely on and are partially only enabled by IT-systems. The fact that business demands change of IT-systems can be observed in all facets. New systems like the ones for self-boarding have been introduced to reduce cost for required staff at the boarding desk [LSY11b]. In the airline industry, acquisitions of smaller airlines are quite popular, which leads to large organizations that have to harmonize and integrate their IT-systems for saving effects. Regulatory requirements in particular are a tough challenge, originating in governments worldwide or being provided by an organization like IATA (International Air Transport Association) [IATA]. Exemplary are the rules about the exchange of passenger information provided by the United States [EPIC]. If an airline's IT-systems are flexible enough they can save the airline a lot of money for changes and provide the airline with competitive advantages by being early on the market.

Software Engineering

Software is the key part of IT-systems that allowed for their big success over the last decades. Software development is a critical and costly activity which has a professional foundation in the discipline Software Engineering [Som07]. The mission of Software Engineering is the construction and maintenance of large-scale software systems with predictable and adequate quality and cost. Software engineering has to cope with increasing complexity, which results from the inherent complexity of the systems being built, the increasing interconnection of systems and integration with existing systems, the continuous change of systems, and from the collaboration of development teams in complex situations [KKN11].

Software Architecture

In order to help control the increasing complexity, Software Architecture has emerged as an important discipline in Software Engineering since the early 1990's [PW92, GS94, SG96] and is still strongly improving [BCK03, TMD09, RW05]. Software Architecture allows to use appropriate abstractions to put order on complexity and get complexity under control. Software Architecture aims at early reasoning and prediction of properties of systems under development in order to get important design decisions right and avoid late and expensive rework [KKN11]. While this describes Software Architecture as a discipline, Software Architecture is also something inherent of any software system. In that sense, it comprises the most important [TMD09] and often hard to change [Fow03] decisions made about a system.

Architecture determines flexibility

Flexibility is an important property of IT-systems and thus also software has to be flexible in order to follow changes of business. Although software is often expected to be easy to change (as it is "soft"), practice shows us the opposite. One main reason for that is that changes often affect key design decisions made, which has far-reaching consequences and is thus costly. That is, flexibility to react on changes strongly depends on a system's architecture and the design decisions made there. Consequently, Software Architecture as a discipline has the responsibility to

	2010	2009	2008	2007
Increasing flexibility	29%	27%	23%	28%
Optimization of processes	21%	21%	-	-
Reduction time-to-market	16%	14%	15%	-
Increasing degree of innovation	10%	8%	9%	9%
Increasing customer satisfaction	5%	3%	13%	13%
Reduction cost	5%	5%	11%	15%
Increasing productivity	2%	7%	14%	13%

Table 1: SOA Check 2010: "Which strategic goals does your company aim at with SOA?" [MER10]

build, among others, flexibility into a system by making the right decisions [CN10, Naa11].

Trends with flexibility potential

Following the need for flexibility, many recent paradigms, trends, and technologies (like SOA, EDA, BPM, BRM²) for information systems explicitly address flexibility [AH06]. Architecturally, these approaches come with architectural mechanisms and technologies that have the potential to construct flexible systems.

High expectations

Due to this inherent flexibility potential and many marketing activities of tool-vendors and consulting companies [IBM06], practitioners expect the resulting systems to be flexible and see this flexibility as one of the biggest advantages of the approaches [GS06]. For example in the study "SOA Check 2010" [MER10], "Increasing Flexibility" was ranked the most important strategic goal (29%) which companies aim at when introducing SOA. Not only in 2010, also in 2007, 2008, and 2009, this was the top-ranked strategic goal (see Table 1).

Missing flexibility in practice

Nevertheless, practice shows that many of today's IT-systems following SOA or other paradigms are not as flexible as expected when looking at the propositions of the paradigms.

Why?
What to do?
↓

This thesis!

In this thesis, we analyze why there are problems with flexibility in practice and why the state-of-the-art does not solve them. Based on this decomposition of problems and reasons, we provide an engineering approach that allows to make use of the flexibility potential of today's paradigms, trends, and technologies and to turn it into true flexibility.

² SOA: Service-Oriented Architecture | EDA: Event-Driven Architecture | BPM: Business Process Management | BRM: Business Rule Management

In practice, business is obviously not always faster than IT. Rather IT often would allow much faster changes than business can do due to all organizational, legal, and social issues. However, business is the leading entity and thus it is worth-while to focus on the best possible support by IT.

1.2 Research Method

As a starting point of this thesis, we motivated that information systems in practice often are not as flexible as expected and we mentioned some problems this fact might cause. In this section, we outline the research method that has been applied for addressing this industry problem. The approach is also illustrated in Figure 2.

Problem statement

The industry problem described is identified in several architecture consultancy projects that Fraunhofer IESE conducted with customers from industry. Additionally, the problems and the background are confirmed by a review of articles on the state-of-the-practice. In order to work towards a solution, we analyze the industry problem and identify main reasons causing it. With the help of these reasons we are able to state more detailed goals for improvement. We describe the problem statements and their decomposition in Section 1.3.

Solution ideas

To achieve the improvement goals, we first select promising research directions based on general ideas from software engineering and soft-

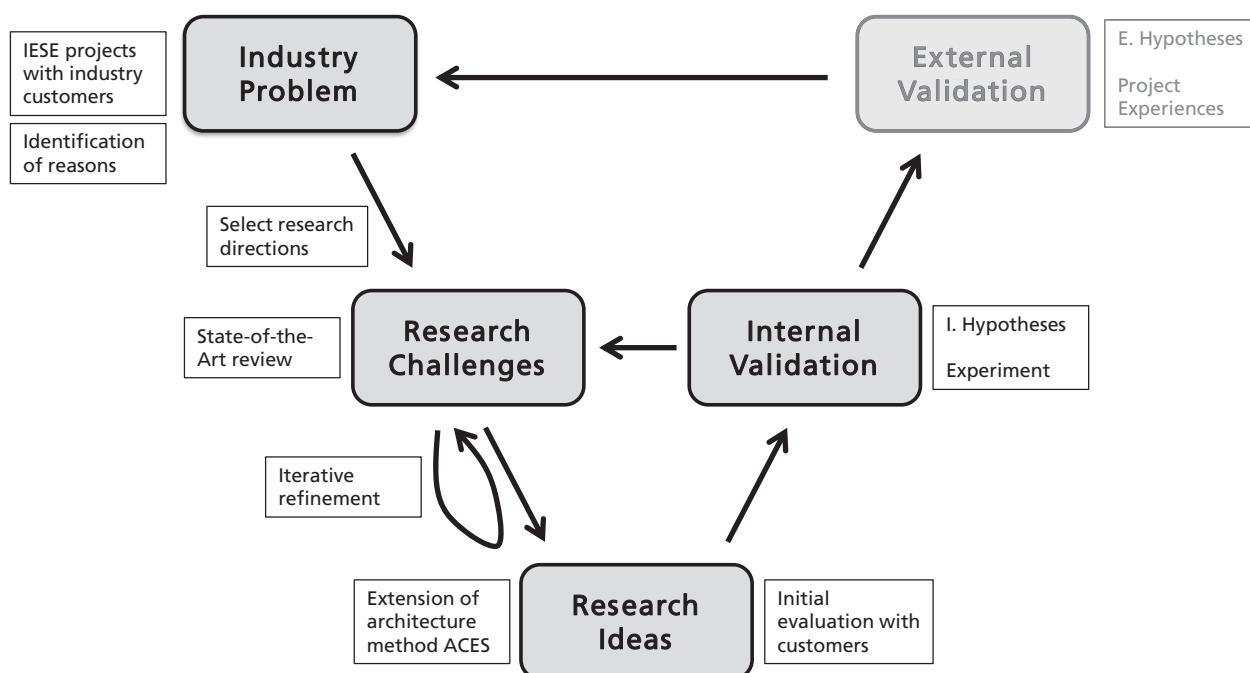


Figure 2: Research method of the thesis

ware architecture in particular. With the help of these solution directions, we check the current state-of-the-art and identify a set of derived scientific software engineering research challenges. Solutions to these research challenges are expected to contribute to the solution of the industry problem. The research ideas to the research challenges can be classified into different categories that are typical in software engineering. First, there are foundational aspects like terminology definition and model building. Second, there are methodical aspects that introduce new approaches and procedures as to how to improve certain engineering activities. Third, there are tool aspects that are necessary to enable practical applicability and scaling of the methods to industry-size engineering projects. All these solution aspects are based on and integrated into Fraunhofer IESE's architecture method ACES (Architecture-Centric Engineering Solutions). We applied the solution ideas early and partially in projects with our customers and iteratively refined the research challenges and solution ideas. We outline the key research directions, derived research challenges, and the research ideas in Section 1.4.

Hypotheses We formulate hypotheses as a prerequisite to validate our solution ideas [WRH+00, ER03]. At the level of the research challenges, we formulate internal hypotheses, which we partially evaluate in experimental settings. At the level of the industry problem, we formulate external hypotheses which are supported by project experiences. We present the hypotheses in Section 1.4.

We explicitly describe the scope and context of the contributions and summarize the key assumptions made in Section 1.5. All research contributions are summarized in Section 1.6.

1.3 Problem Statement

According to our research method described in Section 1.2, we summarize in this section the industry problem, analyze the reasons for missing flexibility and derive research challenges to be solved in order to overcome the problems in practice.

Industry Problem

As motivated in Section 1.1, missing flexibility is an important practical problem of today's information systems. Even systems following paradigms like service-orientation which offers concepts for flexibility and is expected to lead to flexibility, are often not as flexible as expected. We summarize these industry problems (I.P) in problem statements and confirm their relevance in a more detailed discussion.

Statement **I.P1:** Information systems based on Service-Oriented Architecture are, in practice, often not as flexible as needed.

I.P2: The flexibility potential of the paradigm Service-Oriented Architecture is, in practice, often not exploited.

Relevance In order to confirm the practical relevance of the industry problems identified, we look at two important aspects of the problems: frequency and severity.

Frequency: The number of organizations using SOA as a paradigm to organize their IT-systems has continually been growing over the last years. The survey SOA Check 2010 [MER10] reports that the number of enterprises (participating in the study) using SOA increased from 31% in 2007 to 63% in 2010. Table 1 also describes that flexibility is the most important strategic goal of enterprises when adopting SOA for their IT-systems. At Fraunhofer IESE, we recurrently observed in projects with customers from industry that the flexibility of SOA-based information systems was not as good as expected (changes took more than three times as long as expected) and as it would be possible. Also in several research projects that applied SOA as a key architecture style for building software systems, we came to the conclusion that the systems were not as flexible as expected. Still, there is an extremely large number of legacy information systems in use in enterprises. The increasing business pressure on enterprises to integrate their IT-systems will lead to a further growing adoption of SOA or similar paradigms [GBD08, MER10]. Thus, it is important to appropriately apply the paradigm at an early point in time to achieve the flexibility needed.

Severity: Conducting a particular change in an IT-system with ideal flexibility is possible with little effort and time. On the other end of the spectrum, changes are possible that are so massive and have far-reaching effects that the resulting cost is similar to developing the system new. Typ-

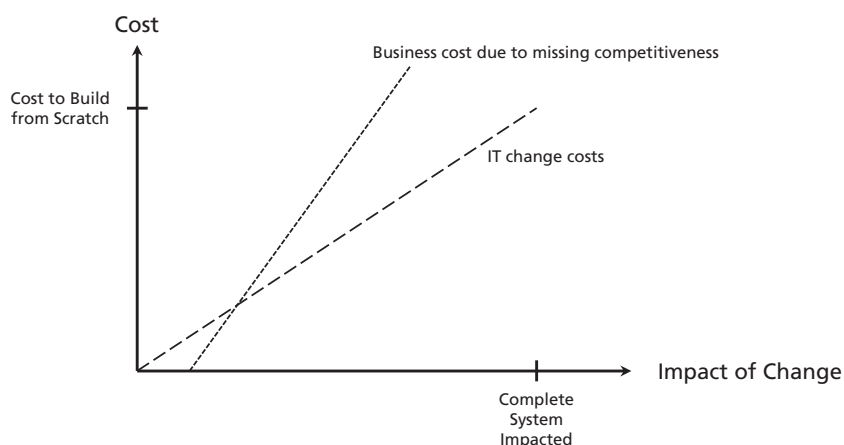


Figure 3: Cost in IT and business caused by missing flexibility

ically, the cost for changes is of course not that high, but can easily cost several 10.000 or even 100.000 EUR and take months to even years, depending on the IT-system and the change at hand. In the overall life-cycle of a system, maintenance causes a large amount of the overall cost [Fri09]. Changes become particularly expensive when they lead to architectural mismatches that have to be resolved [GAO09] or when they affect aspects of the system that require crosscutting solutions [Moo08, Fri09]. At the first glance these costs are costs for a development organization or an IT department of an organization only. However, as described in Section 1.1, there is this strong dependency of business on IT. This means that missing flexibility leads to deferred changes of IT systems and thus to missing or delayed support for business, which can cause higher cost than changing the IT systems (see Figure 3). Awareness has increased that time-to-market or speed in software industry are absolutely crucial for business success [Bos10, KL10]. This speed of change is prevented by inflexible IT-systems and thus the impact of missing flexibility is much higher than only the cost for changing the IT-systems. Missing flexibility of IT-systems supporting key business processes causes immediate competitive disadvantages, which might cost millions of EUR.

Goals

In accordance with the industry problems described we formulate the following industry level goals to be addressed in this thesis.

I.G1: Support architects in constructing SOA-based information systems with improved flexibility.

I.G2: Support architects in better exploiting the flexibility potential of architectural mechanisms provided by the SOA paradigm.

On the way towards achieving G1 and G2 it is crucial to reveal reasons why SOA-based information systems are not as flexible as expected and as the architectural mechanisms of SOA would allow. Thus, we describe in the following which reasons can be found leading to the problems described. Based on these reasons we can refine G1 and G2 into research challenges.

Identification of Reasons for Industry Problems

We stated that an identification of reasons for missing flexibility in practice, despite appropriate architectural mechanisms that are in place, is necessary. Therefore, we first characterize the situation in which the problem arises.

Situations in which flexibility problems arise

Typically, missing flexibility is not discovered when a system is built. It is rather discovered when changes have to be conducted in already finished system parts, an existing system, or an existing landscape of systems. We assume a software developer who has to conduct a certain change to a system or landscape of systems originating in a demand of

business. This developer has to find out where the change has impact and which parts of the implementation have to be changed. If the changes cause hard effort, the system is said to be inflexible with respect to the change at hand. This is reflected in Definition 1, which is rather an intuitive definition. We will introduce a more formal definition in Chapter 4.

Definition 1 Flexibility (Intuitive)

Intuitively, flexibility is the degree to which a system supports a set of anticipated changes to its requirements. [adopted from CK06, Naa09]

Life-cycle aspects

Obviously, the problem of effort and cost intensive changes is not caused in the situation when the change is conducted but when the system has been built or maintained which was at an earlier point in time in the life-cycle of the system.

Focus:
product /
architecture

There are different potential sources of effort and cost intensive changes, which can be classified according to product aspects, process aspects, and organizational aspects [LSR07]. Product aspects typically manifest in inadequate architectural decisions that cause widespread impact of the changes. Process aspects can express in adequate guidance to changes and expensive manual rebuilds of the system. Organizational aspects can express in unclear responsibilities for system parts which delay the realization of changes. However, the product is the key part in achieving flexibility and it has to be appropriate as a foundation for the other aspects. Thus, we focus on architectural aspects of flexibility in this thesis, only. Definition 2 gives a definition of software architecture that will guide our further analysis of the problem.

Definition 2 Software Architecture

"A software system's architecture is the set of principal design decisions made about the system." [TMD09]

Sets of changes as basis for analysis

For the further analysis of flexibility problems, we classify all types of potential changes to a system in a set notation [Naa11]. The classification is mainly along two questions: Are the changes demanded by stakeholders? Are the changes possible according to the architectural decisions made about the system? As a prerequisite, we introduce definitions in order to allow the classification.

Definition 3 Flexibility Requirement

A flexibility requirement is a requirement that expresses the potential need for changing the set of requirements of a software system in the future.

Definition 4 *Architecture Mechanism*

An architecture mechanism can be any type of architectural style, pattern, tactic, etc., which is introduced in architecture design in order to address requirements. Architecture mechanisms are often realized by infrastructure technologies, which means that using such a technology means to introduce the respective architecture mechanism.

Flexibility is facilitated by architectural mechanisms that allow changes to be conducted with as local and little effort as possible [Naa11]. For a given architecture of a system, it can be analyzed whether a certain change can be done with low effort, that means whether the system is flexible with respect to the change. Then, we say the change is in the flexibility potential of a system.

Definition 5 *Flexibility Potential*

The flexibility potential of a system is the set of all potential changes to requirements that can be realized with acceptable effort.

This leads to a very important point: Only having the right architectural mechanisms (see Definition 4) in place in a system does not guarantee that flexibility is really achieved. For example, the architecture decision to organize the business logic along a uniform structure of services with clear interfaces is a good supporter of flexibility, but without knowing how the business logic is mapped (see Definition 6) to the services it cannot be decided whether the system is flexible with respect to particular change requirements. We demonstrate this significant difference in the upcoming example.

Definition 6 *Business Logic Mapping (BLM)*

Business Logic Mapping denotes the mapping of business logic to architectural element types, which are defined by a general architecture meta-model or introduced by the application of particular architectural mechanisms.

Consequently, we introduce a differentiation between 1) the flexibility potential that would be possible with the architecture mechanisms selected and applied in a system and 2) the flexibility potential that remains after the business logic mapping has been done. The latter is by definition a real subset of the first one. This distinction is highly relevant for our characterization of flexibility. Figure 4 illustrates the resulting sets of potential changes and is explained in detail in the following. Below, we use the example from the airline domain to illustrate the different types of changes and the associated problems.

- (0) *All potential Change Requirements*
As described above, we start with the (hypothetic) set of all potential change requirements to a system. Please note that this set cannot be written down; it is a mental support for the classification.
- (1) *Flexibility Requirements*
The set of change requirements as defined in Definition 3.
- (2) *Flexibility Potential of Architecture Mechanisms*
The set of change requirements that can be done with low effort in a system under the assumption that an appropriate BLM was done. This is a hypothetic set, which is a mental support for the classification.
- (3) *Flexibility Potential Considering BLM*
The set of change requirements that actually can be done with low effort in a system. This is the actual flexibility potential as described in Definition 5.
- (4) *Matching Flexibility Potential and Requirements*
The set of change requirements that are on the one hand demanded (flexibility requirements) and on the other hand also in the actual flexibility potential.
- (5) *Flexibility Requirements Missed due to BLM*
The set of change requirements that is demanded as flexibility requirements and could be covered with the flexibility potential of the architecture mechanisms, but which is missed due to the chosen BLM. This is one key reason why SOA-based systems are in practice often not as flexible as expected, because BLM is often not seen as an architectural task and thus not done with the flexibility requirements in mind.

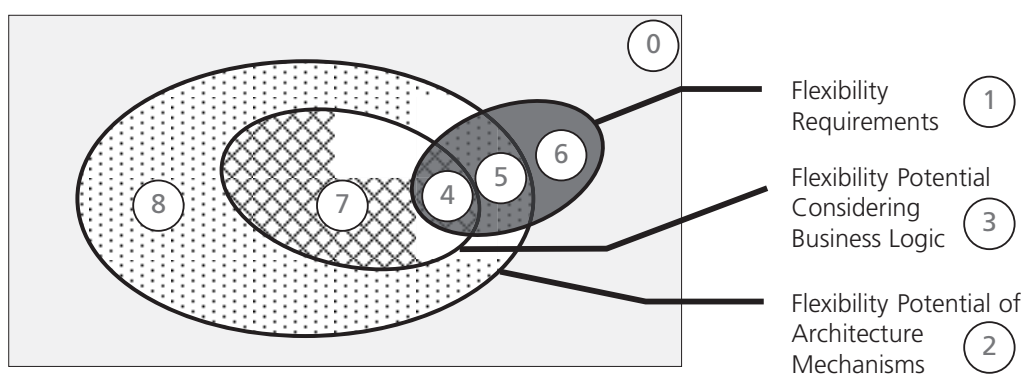


Figure 4: Set representation of requirements around flexibility

- (6) *Flexibility Requirements Missed due to Missing Arch. Mech.*
The set of change requirements that is demanded as flexibility requirements but that is not addressed by any architecture mechanisms providing the necessary flexibility potential.
- (7) *Flexibility Potential beyond Requirements (after BLM)*
The set of change requirements that is provided as flexibility potential by a system but that is not demanded as flexibility requirements.
- (8) *Flexibility Potential beyond Requirements (Arch. Mech. only)*
The set of change requirements that could be provided as flexibility potential based on the given architecture mechanisms, but with a different BLM. However, the requirements are not demanded as flexibility requirements. This set is not of much relevance and only included for reasons of completeness.

Example from airline industry	To illustrate the key points described above, we come back to an example from the airline industry. Please note that the example is strongly simplified with respect to the functionality described and to the detail of architecture modeling. It is restricted to the aspects needed to explain our points related to flexibility. The flexibility requirements are also simplified in the sense that they all focus on business process aspects, which allows keeping the architecture modeling as simple as it is. Of course there are also other types of flexibility requirements which will be introduced in Chapter 4.
Functionality	The example at hand is a CheckIn system that allows running a process with passenger identification, seat assignment and baggage handling (see Figure 5a).
Architecture	<p>The following architectural decisions have been made and are also manifested in architectural views shown in Figure 5b:</p> <ul style="list-style-type: none">• Service-orientation is used to organize the business logic (Architecture mechanism)<ul style="list-style-type: none">○ Services provide encapsulated business logic○ Separation between services and business process• Business processes are realized with descriptive process definition, which is executed by a BP Engine (Architecture Mechanism)• The user interface is automatically generated for business process steps by an UI Engine and can handle sequential handling of process steps (Architecture Mechanism)• Three services are provided: Identify, Seating, Baggage (BLM)• One process is defined with the sequence: Identify, Seating, Baggage (BLM)

- A passenger is the key entity; seats and baggage items are assigned to passengers

Having the architecture designed with architectural decisions as described, we now look at flexibility requirements and how well they are supported by the architectural decisions.

Flexibility requirements

We illustrate with different flexibility requirements (all belonging to set (1)), what makes them fall into the sets (4), (5), and (6).

FR1: Change in the CheckIn process the order of Seating and Baggage.

This change should be easily possible by adapting the declarative process definition, as Seating and Baggage don't have a correlation, they are only correlated to the identified passenger.

→ Matching flexibility requirement and flexibility potential (4)

FR2: Change in the CheckIn process the order of Identify and Seating.

This change is not easily possible as Seating needs a passenger to which a seat can be assigned. That means to conduct this change would mean to change at least the Seating service and the way how a seat can be first reserved and then assigned to a passenger after the identification. In principle, this change requirement would be possible with the architecture mechanisms, but the BLM prevents it.

→ Flexibility Requirement Missed due to BLM (5)

FR3: Change the CheckIn process in a way that Seating and Baggage can be worked on in parallel on the same screen.

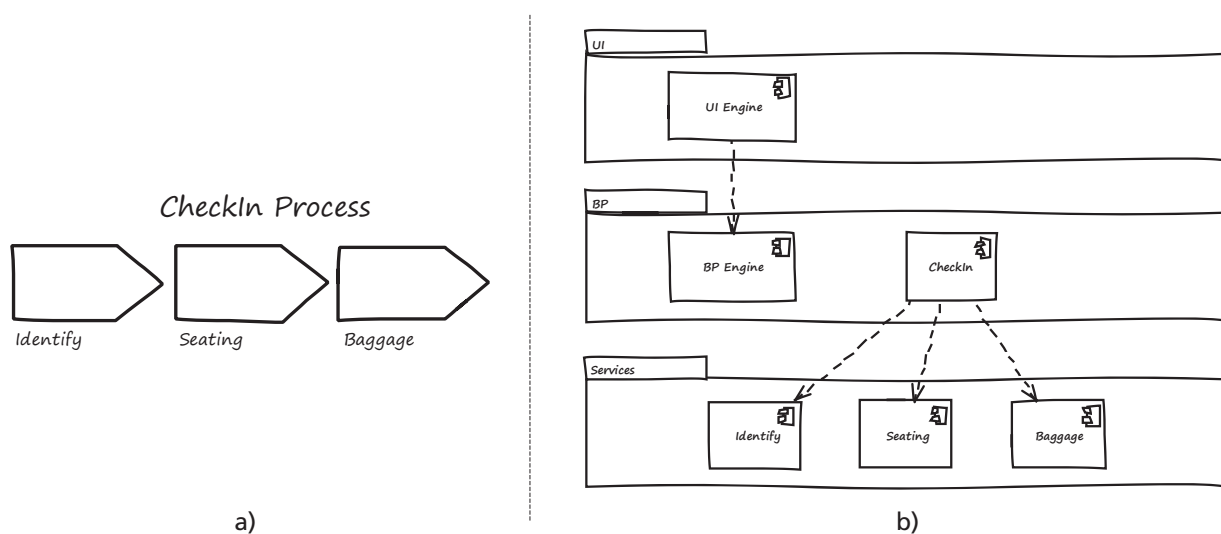


Figure 5: a) CheckIn process b) Simplified architecture of airline system

This change is not easily possible as the UI Engine only supports sequential processing of process steps. Conducting that change would at least require a significant change to the UI Engine and to the interplay with the BP Engine. That means that the appropriate architecture mechanisms to support this change are not in place.

→ Flexibility Requirements Missed due to Missing Arch. Mech. (6)

This example illustrates the key sets of change requirements as introduced in our model ((1), (4), (5), (6)). The other sets are rather hypothetical and cannot be enumerated, but they are very helpful to construct and understand the model. In the following, we will summarize the results from this classification and the derived reasons for missing flexibility.

Analysis Summary and Resulting Goal Hierarchy

Analyzing the reasons for missing flexibility can be done best from the perspective of flexibility requirements. In the previous section, we separated flexibility requirements in three subsets. Based on these three subsets, we can highlight reasons for missing flexibility now.

(6) *Flexibility Requirements Missed due to Missing Arch. Mech.*

In this case, a flexibility requirement has explicitly not been addressed, either because it was not known or because it was deliberately dropped due to some architectural decision making, or because some design mistakes happened.

(5) *Flexibility Requirement Missed due to BLM*

In this case, a flexibility requirement is not in the flexibility potential although architectural mechanisms are in place that would allow for it. However, the chosen BLM prevents covering the flexibility requirement by the flexibility potential. This case, which is often experienced in practice, happens when developers believe they would directly achieve flexibility by applying particular architecture mechanisms, which is not true as shown above.

(4) *Matching flexibility requirement and flexibility potential*

In this case, a flexibility requirement is covered by the actual flexibility potential of a system, resulting from the architecture mechanisms and the BLM. This is the desired case for all flexibility requirements. In order to express this explicit match of flexibility requirements and flexibility potential, we introduce the term "True Flexibility" in Definition 7. Typically, when one speaks of the flexibility of a system, this is exactly what True Flexibility means, namely the flexibility with respect to anticipated flexibility requirements.

Definition 7 *True Flexibility*

True flexibility denotes the set of matching flexibility requirements and actual flexibility potential.

Figure 6 (left) depicts True Flexibility as a real subset of the Flexibility Requirements, as often found in practice. The goal to be addressed in this thesis in order to support the industry level goals I.G1 and I.G2 is to, visually speaking, move the Flexibility Potential over all Flexibility Requirements and thus achieve True Flexibility for all Flexibility Requirements. This is formulated as I.G3, which can be decomposed into two sub-goals.

I.G3: Improve the degree to which a system’s flexibility potential covers its flexibility requirements. (Point in time: Over full life-cycle of system)

Derived from Figure 4, it can be seen that the sets (5) and (6) have to be reduced or eliminated. Reducing set (5) means to improve the BLM for the given flexibility requirements. Reducing (6) has two aspects: First, already anticipated requirements have to be addressed with appropriate architecture mechanisms and BLM. Second, the anticipation of flexibility requirements at the point in time when the system is constructed can be improved. The following two sub-goals express these aspects.

I.G3.1: Improve the degree to which a system’s architecture mechanisms and BLM cover anticipated flexibility requirements. (Point in time: When the system is constructed / maintained)

I.G3.2: Improve the degree to which flexibility requirements are appropriately anticipated when a system is constructed / maintained.

Figure 7 illustrates the relationships among the industry problems and the industry goals as described in this section.

1.4 Solution Ideas and Hypotheses

After defining and focusing the problem domain in the previous section,

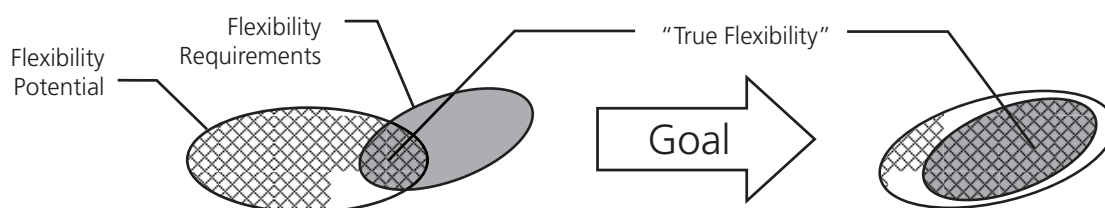


Figure 6: Illustration of true flexibility

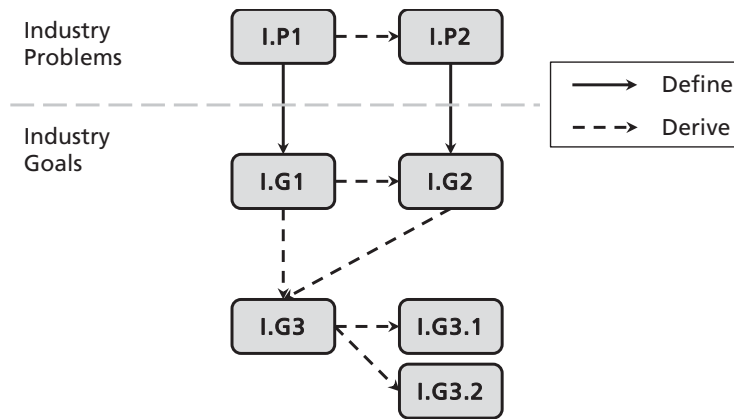


Figure 7: Derivation of industry goals

we now concentrate on necessary research to address the identified goals. Practical problems can potentially be addressed with many different solution ideas resulting in completely different contributions. Therefore, we first outline research directions, the initial ideas how to address the goals. Based on these research directions (R.D), we outline research challenges (R.C) not covered by the current state-of-the-art. To solve these research challenges, we sketch our research ideas (R.I) and finally state research hypotheses (R.H) about expected improvements.

Research Directions

We describe research directions (R.D) as general ideas, where research contributions should be made in a way that the industry goals stated before can be addressed. We introduce a separation of research directions: First, we consider it necessary to extend architecture design methods in general to give better support for flexibility, independent of SOA-specifics. Second, we see the incorporation of SOA-specifics as a promising direction to guide architects even better towards the exploitation of the provided flexibility potential.

General
architecture
aspects

As a basis for methodical contributions, we need a clear foundation of concepts and terminology.

R.D1: Clarify theoretical foundation of relationship between flexibility and architecture.

Improving the architecture design process with respect to a specific property of the system under design, in our case flexibility, can be done with constructive and analytical support. Constructive means to give the architect more specific guidance in the design process in order to achieve the system properties. Analytical means to give the architect feedback on the level of achievement of the property in order to allow quick rework cycles. We include both variants as research directions.

R.D2: Enhance architecture design processes to guide architects towards alignment of architecture mechanisms and BLM with respect to flexibility.

R.D3: Move flexibility measurement from a manual and often neglected task to an automated solution, which allows continuous feedback about a system's flexibility to the architect.

SOA-specific aspects

After equipping the architecture design process with specific support for flexibility, we also want to benefit from the fact that the class of systems we deal with is restricted to SOA-based information systems.

R.D4: Integrate SOA-specific architecture mechanisms with the architecture design process for optimized exploitation of flexibility potential.

Research Challenges

Each of the described research directions bears research challenges. In the following, we describe the research challenges (R.C) addressed in this thesis. They are directly mapped to the research directions, as also depicted in Figure 9. For each research challenge we describe very briefly the gap to the state-of-the-art, which is presented in detail in Section 3.

R.C1: How can the relationship between flexibility and architecture be precisely characterized and how can this be used for 1) better elicitation of flexibility requirements, 2) more guidance for architecture design, 3) measurement of flexibility?

In the state-of-the-art, the relationship between flexibility and architecture is characterized from many perspectives, in particular in the context of evaluation of flexibility. We have to synthesize a consistent meta-model which serves all aspects from flexibility requirements over architecture to the concrete realization at code-level. In particular the constructive aspect is missing in related work, which has to be extended.

R.C2: How can an architecture construction process support architecture design for flexibility with appropriate architectural mechanisms and business logic mappings?

In the state-of-the-art, flexibility as a quality attribute is not in the focus of specific guidance in architecture definition approaches. However, architecture mechanisms like styles and patterns offering flexibility potential are widely published. Existing approaches rather focus on functional decomposition or abstract design for quality attributes, but in particular, the combination always stays very abstract and is a gap to be filled by this thesis.

R.C3: How can the flexibility of an architecture under design be automatically predicted for near-time feedback on flexibility to an architect?

In the state-of-the-art, analyzing architectures for flexibility (or maintainability, modifiability, ...) is the best-populated field. Many approaches focusing on different aspects exist, like the estimation of resulting change costs or the judgment of investments into flexibility. However, all these approaches need human involvement for evaluating the impact of changes on the system. We do not provide completely new flexibility metrics but want to come up with an idea as to how the analysis can be automated using metrics similar to the ones used in previous approaches.

R.C4: How should architectural information about paradigms / technologies like SOA be described and used in architecture construction in order to exploit their flexibility potential?

In the state-of-the-art, architecture mechanisms of SOA are often described in much detail, but without pinpointing the flexibility potential. The flexibility potential is mostly rather implicitly assumed. We aim at making this flexibility potential more explicit in order to systematically exploit it during architecture definition.

Research Ideas

For each research challenge, we describe a short summary of the key research ideas (R.I) addressing the challenge. A comprehensive elaboration of the ideas is given later in the thesis.

R.I1: We decompose flexibility and architecture as concepts and clarify the relationships, as already started in Section 1.3. In particular, the role of architecture mechanisms and business logic mapping are central. We reflect that architecture is only an abstraction of implementation and a means to master the complexity. We build a conceptual model describing the relevant concepts connecting flexibility and architecture as a basis for engineering support. We use architectural knowledge to support working with flexibility requirements.

Architects doing architecture design have to make decisions in order to balance and achieve quality attributes. In the architecture design process, we want to better support architects in addressing flexibility.

R.I2: We provide a methodical extension to existing architecture design methods with a specific focus on flexibility. Therefore, we combine aspects of methods concentrating on functional decomposition and as-

pects of methods focusing on quality driven design. We make the combination concrete and guided by relating the process steps to concrete architectural element types (e.g. business and infrastructure elements) which are introduced to support the alignment of architecture mechanisms and business logic mapping. We give heuristics for the process steps.

Architects have to judge whether the architectural decisions made lead to the flexibility potential needed. This needs analysis and due to missing practical metrics for flexibility architects often do not analyze explicitly. Further, an architect has to deal with many different requirements that need addressing with different architectural solutions. Evolving the architecture design might harm already designed solutions for flexibility, which needs recurring analysis. Thus, we want to give architects continuous and automated feedback on the current flexibility.

As we defined flexibility relative to a set of flexibility requirements, the automated measurement of flexibility cannot operate only on an architecture model, but it has to consider the flexibility requirements. Fully formalizing flexibility requirements and putting them into relation to an architectural model does not seem to be a practical approach. Thus, we take a different approach:

R.13: We extend the architecture model to include all information for the automated analysis. This becomes possible by including the information how the architecture addresses the flexibility requirements. Concretely, that means that an architect has to model the impact of a change requirement in the architecture model. We provide appropriate modeling notations to include this piece of information. The architect has to reason about this in any case, now he also puts it explicitly into the model. We extend a modeling tool (Enterprise Architect) to automatically calculate and display the current flexibility of a system under design.

Finally, we improve the exploitation of flexibility potential of SOA architecture mechanisms.

R.14: We analyze the key architectural mechanisms of SOA for their flexibility potential and how it should be used. This information is packaged for usage in the architecture design process.

An overview of all presented research ideas and how they relate to a simplified description of an architecture design process is depicted in Figure 8. We give a more precise description of scope and context in the next section.

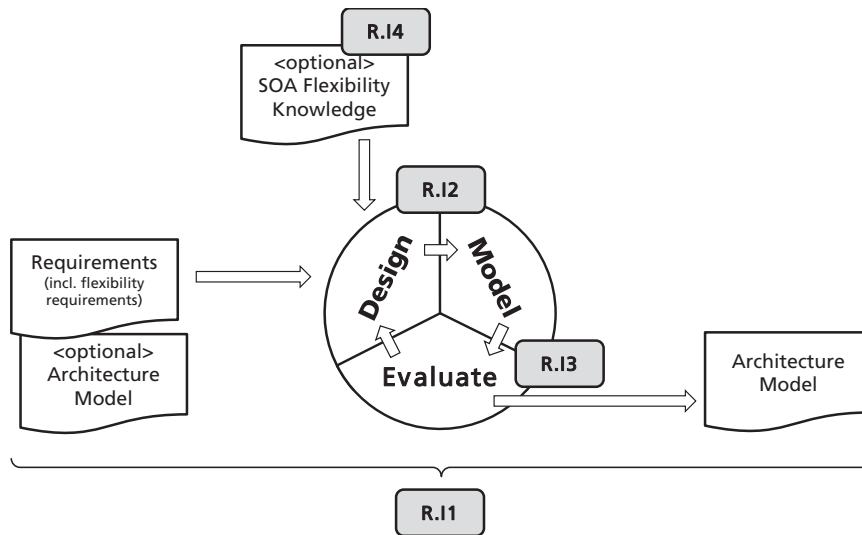


Figure 8: Research ideas in the context of architecture design

Research Hypotheses

In alignment with our research ideas R.I1 (Conceptual Model), R.I2 (Design Support), R.I3 (Evaluation Support), and R.I4 (SOA Flexibility Mechanisms), we define research hypotheses. The research hypotheses address each research idea separately; additionally, we define research hypotheses spanning across the ideas. Our research hypotheses cover the aspects **Validity**, **Effectiveness**, **Efficiency**, and **Applicability**. With our hypotheses, we sketch the space of expected benefits of our contributions (see Section 7.1).

For a concrete and detailed validation, we focus. One aspect of the effectiveness of our flexibility evaluation contribution is the following hypothesis: *“By explicitly describing how a flexibility solution for a particular*

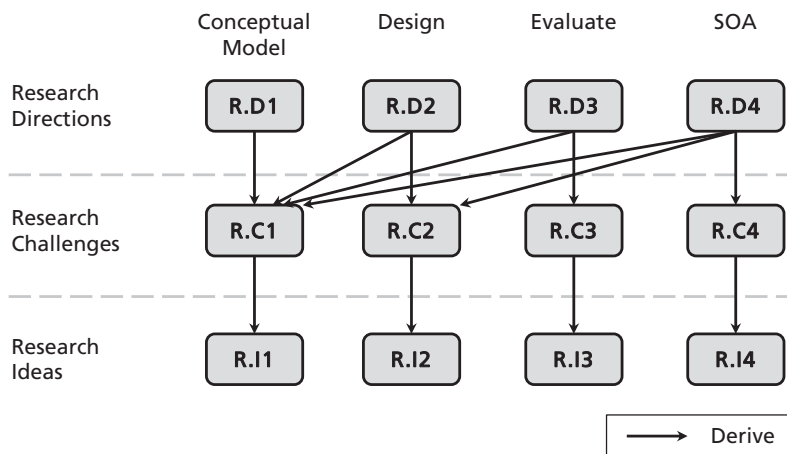


Figure 9: Relationship between research directions, challenges, and ideas

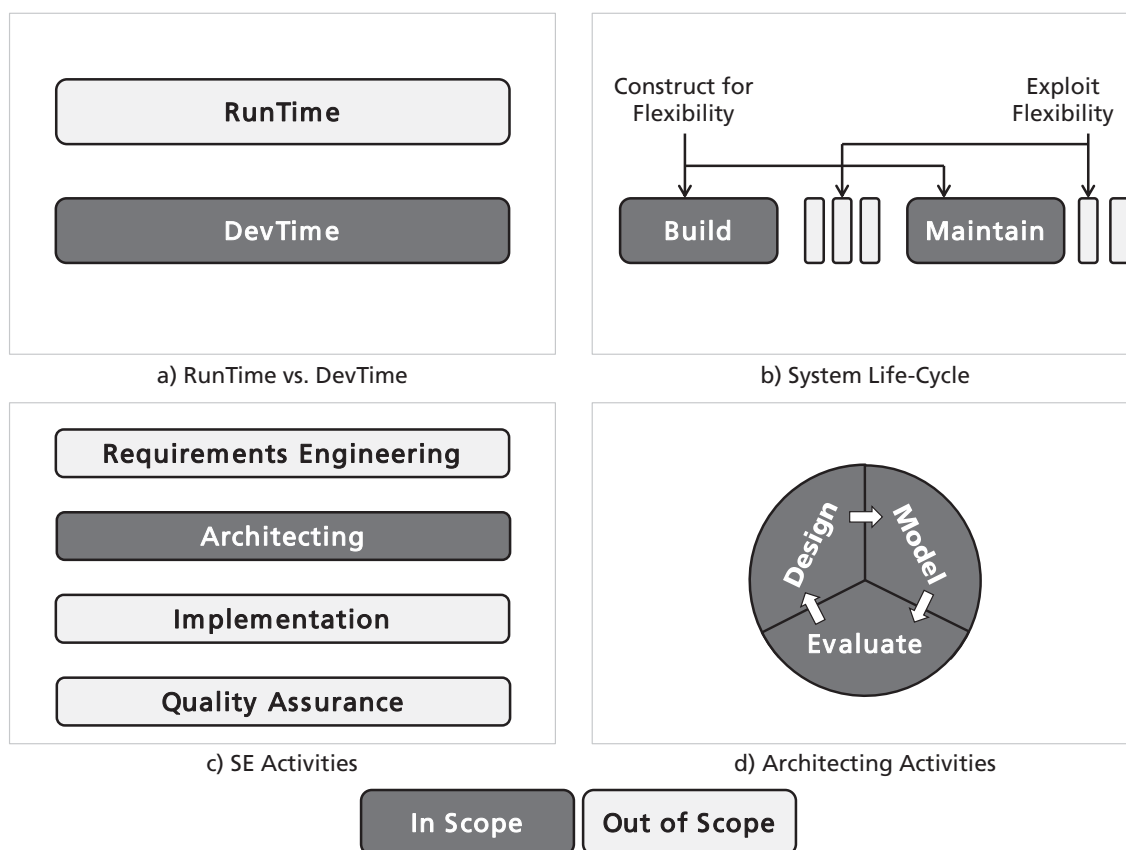


Figure 10: Scope and context of the thesis

scenario works, architects produce more flexible architectures. We further refine this hypothesis and evaluate it in an experiment. In the experiment and in projects with industry we collected further qualitative results for additional hypotheses.

Summary

In this section, we elaborated the ideas how to address the industry problems and goals presented in Section 1.3. The research ideas in summary provide the solution for the goals; thus there is no concrete mapping. An overview of the directions, challenges, and ideas and their relationships is depicted in Figure 9.

1.5 Scope, Context, and Assumptions

The previous section described the ideas underlying this thesis. In order to ease understanding of the ideas and how they contribute to the goals, this section pinpoints the scope of the contributions and thus describes the context and assumptions made.

Business vs. IT	As described in Section 1.1, business demands quick changes of supporting IT-systems. This thesis focuses its contributions on IT , and in particular on software engineering, but of course always aligned with the needs of business.
System types	The practical problem motivated in Sections 1.1 and 1.3 have SOA-based information systems as a background. That is, the contribution is best tailored for this type of systems. It addresses both, single systems and landscapes of systems that are architected with service-orientation. One key assumption when optimizing flexibility is that the systems are under control of the architect . If not, the measurement of flexibility still works and provides helpful insights for potential workarounds. As a large part of our contributions is not system-specific, they are also transferrable to other system types like embedded systems.
Leverages for flexibility	In Section 1.3, we described that architecture, development process, and organizational aspects are all relevant in order to achieve quick reactions on change requests. In this thesis, we concentrate on architecture aspects only, which means we bring the system into a shape that is the foundation of quick reactions on change requests.
DevTime vs. RunTime changes	If changes to the behavior of a system are necessary, these changes can be principally conducted at development time or at runtime. Changes made at runtime are in any case already incorporated into a system and require self-monitoring of the system in order to recognize which configuration of behavior to expose. We focus in this thesis on DevTime changes (see Figure 10a). All flexibility requirements are future but (potentially) anticipated requirements. They have not been realized yet but have to be realized by changes to the implementation of the system. The goal in this thesis is to make such changes as cheap as possible when they have to be realized.
System life-cycle	In the overall life-cycle of systems, the contributions of this thesis are mainly applied during the construction phase (see Figure 10b). During this phase, flexibility is built into the systems targeting at the exploitation of flexibility at later points in time. Building in flexibility is an early investment which pays off later when quick reactions to change requests are necessary. In general, it is expected to be cheaper to build in the flexibility potential than to change a system when the flexibility potential is missing. Also later in the life-cycle of a system there might be phases of major maintenance which can be used to create new flexibility potential for newly anticipated flexibility requirements.
Software engineering activities	As described before, architecture is the key to facilitate flexibility in IT-systems. Thus, this thesis focuses on architecting (see Figure 10c) to achieve flexibility. This thesis does NOT introduce a new architecting method. Rather it proposes methodical enhancements which combine aspects of existing architecture methods and is compatible with many existing methods. However, architecting for flexibility also requires to

look at the requirements level (since flexibility is always about change requirements) and the implementation level (as architecture is only an abstraction of the implementation, which facilitates more efficient decision making; in fact, the key effort of expensive changes always has to be made to change the implementation).

Software engineering roles

The method contributed in this thesis clearly targets at **architects** in software development projects and organizations. Thus, it is a method for a **small group of experts**, which can assume a high level of architectural knowledge as prerequisite.

Architecting activities

With the method contributed in this thesis, architects are guided and supported in key activities (see Figure 10d) of architecture design with a specific focus on flexibility: **Design** and decision making, **modeling** the decisions in a consistent model, and **analyzing** a current draft of the architecture with respect to achieving true flexibility. An underlying assumption is that architecture design, modeling, and analysis are cheap compared to the real implementation. That is valid for both construction and change.

Methods vs. architecture mechanisms

Architecture mechanisms are necessary to achieve flexibility. However, this thesis does not contribute new architecture mechanisms, rather it focuses on **methodical** support and shows how to make best use of them to achieve flexibility.

Automation potential

Based on the ideas described in Section 1.4, the **analysis of architecture models with respect to flexibility** is formally defined in such a way that it can be **automated with tools**. The tool calculates the current achievement of true flexibility. The design and modeling activities are still manual tasks for architects. However, design and decision making is supported in so far that the continuous analysis gives instant feedback as soon as the modeling of decisions is done.

Flexibility as a quality attribute

Flexibility is an **important quality attribute**, but there **are other important quality attributes as well**. Focusing so strongly on flexibility in this thesis does not mean that it is more important than other quality attributes. This **prioritization** depends on the concrete system. The approach we describe for flexibility can be rather seen as a small slice in software engineering cutting across all activities like requirements engineering, architecting, or implementation. In the same manner, further research is conducted or still needed for other quality attributes. In the end, architecture is the point where to balance between competing quality attributes. When all quality attributes and their approaches extend the same architecture model, there can be much better support for tradeoff decisions than there is today.

1.6 Contributions Overview

Contribution list In the previous sections, we sketched the story of this thesis from the problem definition over the selection of research directions down to the concrete research ideas. In this section, we outline the key contributions this thesis makes:

- **Conceptual Model:** We provide a conceptual model that partially formalizes and relates key aspects of flexibility and architecture. It is the foundation for a clearer understanding of what flexibility is and how it is addressed by the further contributions in this thesis.
- **Design method enhancement for flexibility:** We provide constructive, methodical support for software architects targeting at high flexibility in their systems. This methodical enhancement makes design activities more explicit and gives guidance and heuristics with a specific focus on flexibility. It combines aspects from functional decomposition and quality-driven design approaches and leads to well-aligned architecture mechanisms and business logic mappings, targeting at anticipated flexibility requirements.
- **Automated measurement of flexibility in architecture tool:** We further enhance the design process by providing continuous feedback on the current flexibility level of the system being architected to the architect. This is supported by automated measurement which is integrated in the architecture tool "Enterprise Architect". In order to support this automated measurement, architects enrich the architecture model with information on the addressing of flexibility requirements
- **Packaged SOA flexibility mechanisms:** We provide for typical architecture mechanisms of SOA descriptions how they can be applied to contribute to flexibility. We achieve this by sketching for the key architecture mechanisms which flexibility potential they bear and to which typical challenges they contribute.
- **Validation results:** In a controlled experiment we got empirical evidence that explicitly modeling change impact during architecture design helps architects to create significantly more flexible architecture solution.

Contribution categories The contributions of this thesis can be assigned to the following categories:

- **Foundations / Formalization:** Contributions that have basic character and provide the conceptual model the other contributions can built on

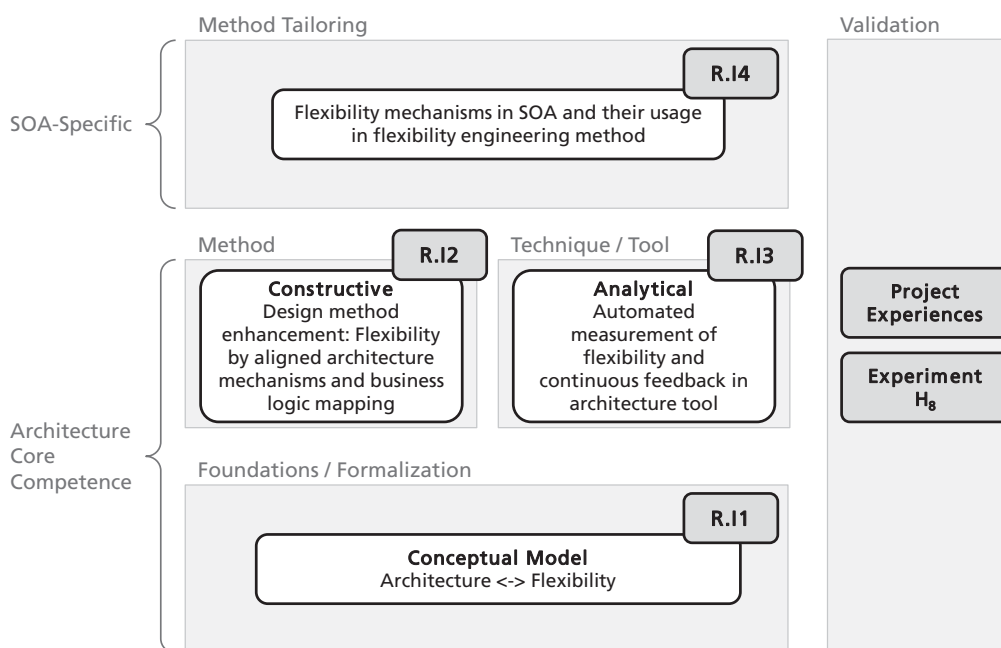


Figure 11: Contributions of the thesis in categories

- **Method:** Contributions that represent or enhance an engineering method, which guide certain engineers in a particular activity in the software engineering lifecycle
- **Technique / Tool:** Contributions that represent a concrete technique like algorithms for measurement. Further, there are contributions that make use of the fact that certain activities in the development process can be fully automated due to their degree of formalization, which can be realized in engineering tools
- **Validation:** Contributions that empirically validate certain aspects of the contributions to check whether the proposed benefits can be achieved

Figure 11 visualizes the relationships between the contributions and the categories described. Further, in this figure we also link the research ideas as described in Section 1.4 to the contributions.

1.7 Thesis Outline

The introduction (i.e. this chapter) motivates this thesis and presents the key ideas. We state the practical problem and analyzed reasons which are used to find research directions. We derive research challenges and sketch the key ideas to address the challenges. In order to precisely describe the scope of the thesis, we put it into different aspects of context and stated the key assumptions.

Chapter 2 continues with a description of foundations for this thesis in the areas of architecture-centric engineering and Service-Oriented Architecture (SOA).

Chapter 3 describes the state-of-the-art, mainly in the areas of flexibility as a quality attribute, architecture design in general and architecture design in the context of SOA. Another closely related field of research is the analysis of quality attributes like flexibility or maintainability.

Chapter 4 describes our conceptual formalization of flexibility as a quality attribute and the particular role of architecture for flexibility. It introduces in particular our metrics for flexibility and the concept of change impact views as additional architecture views. Summarizing, all ideas are put into relation in a conceptual model of flexibility.

In Chapter 5, our engineering method enhancements for flexibility are presented, along with the ideas on continuous flexibility measurement and the realization in Enterprise Architect.

Chapter 6 presents the specific aspects of flexibility in SOA-based information systems. Architectural principles, mechanisms, and technologies of SOA are collected and analyzed for their potential support of flexibility.

In Chapter 7, our empirical validation is described. First, the space of hypotheses derived from our research is sketched. Then, a controlled experiment with the gathered quantitative results is presented. Finally, we describe experiences from projects with customers from industry in which we collected further qualitative results.

Chapter 8 concludes the thesis summarizing the results. We further discuss the achievements and their limitations and sketch future work.

2 Foundations of Architecture

*“Complex problems have simple,
easy to understand, wrong answers.”
H.L. Mencken*

In this thesis, we present an architectural approach towards the improvement of flexibility as a quality attribute of long-living information systems. Whereas we briefly described the scope and the context of the contributions in Section 1.5, we explain in this section the foundations of architecture as a basis of our contributions, and how we integrate the contributions. First, we will focus on architecting as an engineering activity in Section 2.1. Second, we will focus on SOA (Service-Oriented Architecture) as a paradigm of designing information systems in Section 2.2. As we see SOA as a wide-spread and promising candidate for architectural support of flexibility in information systems, we explain the necessary background as needed for our contributions.

2.1 Architecting as an Engineering Activity

Flexibility in software systems is achieved by making the right architectural decisions in order to limit and focus the impact and effort of changes. The activities in software engineering aiming at an appropriate architecture are called architecture design or architecting. Thus, architecting is the activity that we enhance with the contributions of this thesis. Therefore, we highlight the essence of architecting and how it is typically done in practice. As a methodical framework for our contributions, we give an overview on the ACES (Architecture-Centric Engineering Solutions) architecture approach of Fraunhofer IESE.

2.1.1 Definitions and Essence

Definitions Architecture as an artifact or concept in software engineering has many definitions. We present a few very prominent ones to elicit the essence. The definition we use as the leading definition in this thesis (see Section 1.3, Definition 2) is:

“A software system’s architecture is the set of principal design decisions made about the system.” [TMD09]

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.” [BCK03]

“Architecture is a set of concepts and design decisions about structure and texture of software that must be made prior to concurrent engineering to enable effective satisfaction of architecturally significant, explicit functional and quality requirements and implicit requirements of the product family, the problem, and the solution domains.” [JLR00]

Summarizing, architecture is about key decisions of software systems which typically manifest in abstracted structures with elements, properties, and relationships. The last definition also includes a purpose, namely to allow fulfilling key requirements.

Abstraction as key concept Abstraction is a key concept of architecture [TMD09, BCK03, Gor06]. By abstracting from many details, architects can reason about essential design decisions on the level of a complete, large system or even landscapes of systems. This allows also to reason about a system which has not been built yet, as an architecture is much cheaper to build and easier to oversee and to analyze than the real system.

Architecting activities Consequently, architecture is a critical asset in software engineering. Architecture provides support for all life-cycle phases of software systems: construction, maintenance, evolution, migration, and retirement. In order to make this support concrete, architecting activities around architecture are necessary: Understanding requirements, designing architecture, documenting architecture, communicating architecture, analyzing and evaluating architecture, implementing based on architecture, ensuring of conformance of implementation to architecture [BCK03, Bos00].

In the following sections, we present architecting activities as done in practice and we present the ACES approach for architecting.

2.1.2 Architecting in Practice

How architecting is done in practice covers a very broad spectrum regarding the level of sophistication. It ranges from no explicit architecture work at all to sophisticated architecture-based quality predictions with architecture models. We provide a brief overview on architecting in practice, mainly based on our experiences collected at IESE in more than 50 recent architecture projects with customers from industry and based on reported experiences in literature.

Criticality	Recently, the criticality of architecture for successful software development is more and more recognized in industry [CS09, Boo06].
Purposes of architecting	Frequently the architecture's only target is to prescribe the implementation. That means that architecture's power to reason early about later properties of the system under construction are neglected [Boo07]. In particular, there is often no common understanding in companies why architecting is done and what the expected benefits are. This leads to the situation that it is very hard to decide how much to invest into architecting activities. However, there are also cases where companies use their architecture models for analyses like performance predictions or for generation approaches like in Model-Driven Development (MDD).
System decomposition	Architecting in practice often means to only provide a high-level blueprint of the system under construction. Accordingly, as an intermediate step from the requirements to the implementation, the system is decomposed into smaller, manageable pieces.
Modeling and documentation	The resulting architecture blueprints often have only informal semantics and are ambiguous [CBB10]. Typically, they lack precision and prescriptiveness for key architectural aspects and are often represented as "boxes and lines". Architecture modeling with tool-support is rather seldom, in particular explicit architecture models that support particular purposes are seldom.
Quality attributes	Although an appropriate architecture is a crucial factor for achieving key quality attributes of software systems, there is still little focus on these quality attributes in practical architecture work. Quality attributes are mostly stated rather vaguely and the opportunity to analyze the fulfillment of quality attributes with the help of architecture is mostly missed. In particular, development time quality attributes like flexibility are often not explicitly addressed in architecture design. On the one hand, indeterminations and uncertainty seem to make architecting for these quality attributes difficult; on the other hand design methods only give rather limited guidance for architects. If development time quality attributes are addressed this is typically done based on previous experiences of the architect. If development time quality attributes are evaluated this is typically done with informal expert estimations.
Connection to code	Architecture work can be only effective if the architecture is consistently reflected in the implementation. In practice, we often observed that companies do not intensively care about consistency between architecture and implementation [Kno11].
Architects' background	In industry, senior developers often become software architects. Unfortunately, there is no common educational foundation for architects, which is also rarely taught at universities. This results in a situation where

architects even in the same company do not have a common idea and language about architecture and they also do not have a common approach to architecting [Cle10].

Architecture & Agile In recent years, agile development has become highly popular in software companies. The most popular method, Scrum, [SB01] actually does not tell anything about how architecture work should be done in agile projects. Rather, it only states process aspects as to how to realize requirements iteratively. Often this leads to the perception that architecting as an activity and architecture as an artifact are not needed any more in agile development [ABK10, Kru10].

Summary Architecture is being established as a discipline in software engineering also in practice. However, many potential benefits of architecture have still not been achieved due to missing knowledge, missing methodical guidance, and also missing scalability of available approaches. We target in this thesis at providing more methodical guidance for the quality attribute flexibility in architecting and design method and tool support in a form that scales to real-world systems.

2.1.3 The ACES Approach

Fraunhofer ACES (Architecture-Centric Engineering Solutions) is Fraunhofer IESE's approach to architecting [KKN11]. ACES is not intended to be a completely new architecture method. Rather, it integrates aspects of several existing architecting approaches [BCK03, Bos00, RW05] and focuses on strong practical applicability, as the overall idea of Fraunhofer proposes. Besides integrating and tailoring existing approaches, ACES also adds completely new concepts like the so-called architecture engagement purposes or a separation in architecture core competence and domain competence. ACES has been applied in dozens of projects with industrial customers and is continually evolved with the aid of practical applications and IESE's research contributions around architecture.

ACES [EKK+10, KKN11] is the methodical foundation of the contributions of this thesis.

Architecture Engagement Purposes

Investment and benefit In practice, the value of architecture work is often hard to grasp and hard to demonstrate. As architecting always means to spend effort it is important for the activities to clearly target at goals and do not become an end-in-itself. Generally, architecture serves two main goals: Supporting decision making and realizing the decisions. The respective decisions can be all types of decisions around IT: investment, personnel, technical, organizational decisions. Summarizing: Architecture work is an invest-

ment to achieve particular benefits around software development or usage.

AEP In ACES we introduced the concept “Architecture Engagement Purpose” (AEP) which describes a clear purpose why a particular company does architecture work. We provide a template for describing an AEP in more detail, containing for example the stakeholders and perspectives involved, the purposes and questions, the context, and the expected results of working with architecture. AEPs guide architects in justifying why architecture work is needed, how much has to be invested in architecture, which level of detail is needed, and in deriving the architecting activities that have to be performed.

AEP classes We identified three classes of AEPs that contribute to the goals of decision making and realization (see Figure 12):

- **Prediction:** A key purpose of architecture is to analyze and predict certain properties of software systems. The reason why this is done with architecture, and not for example with the implementation, is that a system’s architecture allows focusing on the essential aspects of a system for the particular prediction. During system construction, an architecture can be designed at much lower cost than the implementation. After the construction of a system, architecture provides the advantage to concentrate on the relevant aspects and thus cope with the often high complexity of the implementation. The key principle of architecture allowing these advantages is abstraction. Architecture abstracts from many details and thus allows concentrating on the relevant aspects for an analysis or prediction. Prediction AEPs mainly serve the goal of decision-making. Prediction is used directly in the construction of architectures in order to make the right architecture decisions or later to check the validity of decisions.
- **Derivation:** A second key purpose of architecture is the derivation of other artifacts. Architecture can be seen as a blueprint of an implementation and has a prescriptive nature for all implementers. Not only the implementation can be derived from architecture; it could also

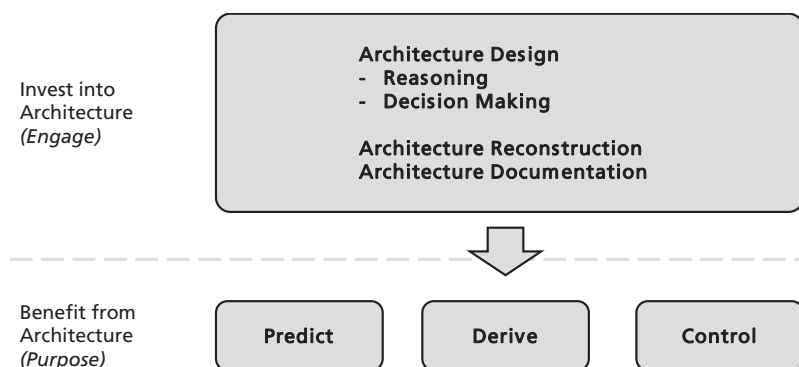


Figure 12: Architecture engagement purposes

be project plans, test cases, etc. For derivation, architecture has to be well described and understandable for the respective stakeholders. Derivation AEPs mainly serve the goal of decision-realization.

- **Control:** A third key purpose of architecture is to control whether artifacts derived from the architecture are compliant with the architecture. The effort invested in designing an architecture and making the predictions about properties of the resulting system only pay off if the actual implementation also realizes the architectural decisions. However, in practice there are often deviations introduced for multiple reasons [Kno11]. Thus, there are also AEPs targeted at controlling the compliant realization of the architecture, which contributes to the goal of decision-realization.

Explicit
architecture

In order to achieve benefits from architecture, investments are necessary. Architecture has to be available in a form that serves the expected purposes. In particular, architecture has to be made explicit to be useful. Explicit means any type of model or documentation which is appropriate to use the architecture to achieve the purposes. For communication of certain well-known aspects, a rather high-level sketch of the architecture might be enough, whereas a reliable performance prediction requires a detailed architecture model with all information that impacts performance. Although every system has an architecture [BCK03], an implicit architecture does not help achieving the benefits as sketched. Providing an explicit form of architecture is typically done in the activities architecture design (for new or changed architectures) and architecture reconstruction (making implicit architecture explicit again by reconstructing it from the implementation) (see Figure 12). The effort to be invested in these activities mainly depends on the AEPs.

Architecture-Centrism

Architecture
as mediator
among com-
peting re-
quirements

Software systems are intended to support a company's business with technology-based solutions. Further, a software system is the crucial element for another company's business: software development business. Further players might be involved in the operation of the system. This shows that many companies might be connected in some way to a software system, and all have requirements to be fulfilled or impose constraints on the software system. These requirements might be competing; very typically, a software system has to provide a certain quality and should be developed in a certain time for a certain amount of money. In order to allow an appropriate software system to be built, the right design decisions have to be made supporting the requirements in an acceptable way. The architecture of the system is the right place to make these decisions. Visually speaking, architecture is the hub that connects all different sources of requirements and constraints and allows their balanced fulfillment.

Architecting as the hub of development Looking at a typical software development process, architecting is a strongly interconnected activity. Architecting has interdependencies with requirements engineering, the implementation, quality assurance, and project management as such. Architecting makes decisions to fulfill requirements and the system is implemented according to the architecture. In a timeline view, architecting is not a phase at a certain point in time of software development. Due to the strong interdependencies of architecting to the other development activities (in both directions), architecting as an activity is supposed to last for the whole development project in ACES. Of course, there are points in time where more or less effort is spent on architecting. The graphical illustration (see Figure 13) of the Rational Unified Process (RUP) [Kru03] is well suited to represent the facts described.

Architecting as the hub of all life-cycle phases While Figure 13 mainly represents the situation of system development, architecture also plays a crucial role in other life-cycle phases of software, as in the maintenance phase or even the retirement phase. Also then, architecting is necessary to make appropriate decisions about the system and its context and to realize these decisions.

Architecting with ACES

An adequate architecture allows building or evolving a system with the right balance among all requirements of the stakeholders.

Architectural requirement Addressing stakeholders' requirements adequately with architecture requires knowing the requirements in a precisely described form. ACES represents architecturally-significant requirements, in particular quality attributes, as architecture scenarios [CKK01, BCK03, RW05].

Architecture design In particular, quality attributes need crosscutting, uniformly realized architectural solutions. Typically, quality attributes are addressed using architectural styles, patterns, or tactics [BCK03, Bos00, BMR97]. ACES provides an architecture design method involving the functional decomposition of a system and the application of styles, patterns, or tactics to achieve the quality attributes. In ACES, as well as in other architecture

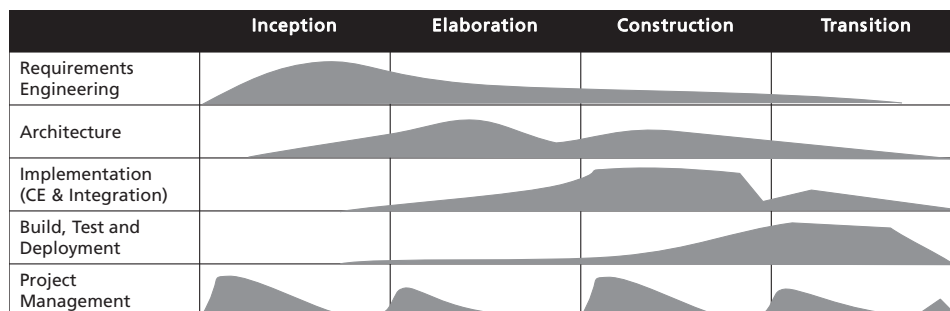


Figure 13: Development phases according to RUP [Kru03]

design approaches, a general process for addressing quality attributes is provided. Additionally, selected sets of tactics supporting particular quality attributes can be given.

Prediction of quality properties

Architecture as an abstraction of the system under construction offers the possibility to predict whether the system will have the intended quality properties before implementing the system at high cost. In the architecture design method, there is a recurring analysis step that checks with predictions of the quality properties whether the architecture is adequate or has to be improved. How precise and automated such a prediction can be done depends on the precision and formalization of the architecture description.

Architecture models and views

In ACES, architectural decisions are manifested in architectural models. The resulting models are the foundation for analysis and prediction, for communication, or for the generation of well-readable architecture documents. As there are typically many different aspects of architecture, as for example runtime structures or development structures, deployments and technologies, or data aspects, ACES also uses the concept of architectural views and viewpoints [CBB10, RW05, HNS99, Kru95, IEEE00]. ACES provides a generic framework (ADF: Architecture Decomposition Framework) that covers all architecturally relevant information and that is project-specifically instantiated to create the appropriate set of architectural views. The necessary information depths and precision depends on the concrete AEPs that should be applied to the architecture model. Ideally, there is one architecture model with the selected views on it, which is the basis for all AEPs. Then it is also possible to determine tradeoffs among quality attributes on the architecture model.

Architecture implementation

An adequate architecture is only of value if it is compliantly realized in the implementation. Thus, ACES provides guidance for the derivation of an implementation from the architecture and further allows the checking of compliance with tool support.

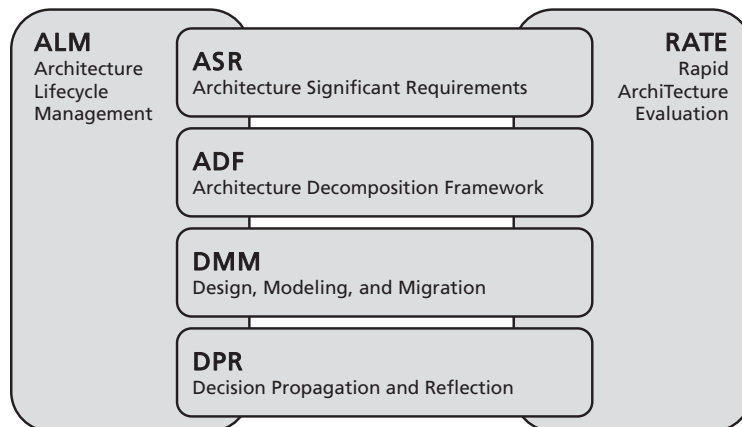


Figure 14: Competence packaging in ACES

Competence Packaging in ACES

Competence areas	All described aspects of ACES are packaged in six competence areas (see Figure 14). ASR (Architecture Significant Requirements) covers all competences around eliciting and representing requirements for architecture work. ADF (Architecture Decomposition Framework) covers all competences around the documentation of architecture with models, views, etc. DMM (Design, Modeling, and Migration) covers all competences around the methodical guidance for making architecture decisions and representing them in models. DPR (Decision Propagation and Reflection) covers all competences around the connection of architecture with implementation and other derived artifacts. RATE (Rapid Architecture Evaluation) covers all competences for evaluating adequacy of architectures and their compliant realization. ALM (Architecture Lifecycle Management) covers all competences around the management of architectural artifacts over time, managing complex models, and aligning architecture activities with other development activities.
Tool support	In ACES, the standard tool for architecture modeling is Enterprise Architect (EA) [EA11a]. ACES reflects the ADF via MDG Technologies in EA and uses the Add-In mechanism [EA11b] of EA for enhanced support of architects. With the help of the SAVE (Software Architecture Visualization and Evaluation) tool [BHS+08, KMN+06, KMN06, KMN08], ACES supports reverse engineering of architecture and compliance checking of intended architectural models with the actually implemented architecture.
Architecture core competence vs. domain competence	ACES as an approach to architecting is independent of particular domains or system paradigms. In order to provide better support for architects of certain system types, it is important to analyze and understand the specific challenges of these systems. This can be in particular common business goals and common requirements like quality attributes.

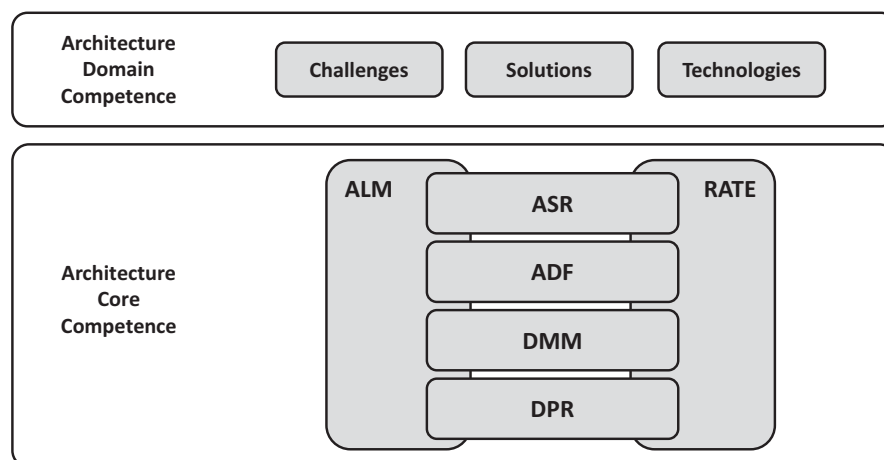


Figure 15: Architecture core and domain competence

E.g., mobile applications often share requirements like offline capability. Based on these common challenges, common solutions can also be provided, typically in form of styles, patterns, or tactics [BCK03, Bos00, BMR97], as described before. Finally, common solutions are often realized in form of reusable technologies (e.g. communication technologies), which are highly relevant for architects to efficiently realize systems. Designing architectures for a particular domain of systems highly benefits from the knowledge of typical challenges, solutions, and technologies [CJM+08]. Figure 15 depicts the overview of core competence and domain competence. In this thesis, the methodical part belongs to the domain-independent core competence whereas the specialization towards SOA belongs to a domain competence.

Contributions to ACES

ACES provides the foundation to integrate the contributions of this thesis. The contributions extend in particular DMM by adding more concrete guidance for designing, modeling, and analyzing flexible architectures. However, our contributions can be also used with other architecture approaches.

2.2 Service-Oriented Architecture

SOA is a paradigm for the construction of information systems, which is widely used today. It comes with mechanisms for flexibility, which in practice are rarely effectively exploited. This led us, as described in Chapter 1, to the motivation for this thesis. As SOA is also a widely used term with many different notions, we provide in this section a brief overview on different perspectives on SOA, how SOA is used in practice, and what it architecturally means to organize a system along the principles of service-orientation.

2.2.1 Definitions and Essence

Not THE definition of SOA

Service-Oriented Architecture as a term was coined by Gartner as early as 1996 [SN96]. In recent years in particular, it was used heavily in IT industry. Nevertheless, there are still many different perspectives on and definitions of SOA available. They are also considerably diverging regarding their level of abstraction, scope, and focus. A set of different perspectives is given in [Til08]. Typical extremes in a spectrum of definitions are very high-level business-oriented definitions like "A paradigm to achieve better alignment of IT and Business with the goal to get more flexibility for business processes." [Til08] and on the other end very technical definitions like "A standardized technical architecture, based on XML, SOAP, WSDL, UDDI, and further WS-* Standards." [Til08]. The first one might be from the perspective of managers with a strong focus on IT strategy and cost savings, the latter one might be from the perspective

of developers developing in a service-oriented environment with some standard technologies.

There are also definitions from standardization bodies like OASIS (Organization for the Advancement of Structured Information Standards): “Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.” [OAS06, OAS09]. Further, there are also books on SOA giving definitions like in [KBS04]: “A Service-Oriented Architecture (SOA) is a software architecture that is based on the key concepts of an application frontend, service, service-repository, and service bus. A service consists of a contract, one or more interfaces, and an implementation”.

SOA as a paradigm

It can be observed that SOA is often seen as a paradigm (which is the broadest scope, covering many sub-aspects like architecture or technologies), whereas other definitions aim at particular sub-aspects like architecture or technologies only. One key aspect that is found in nearly all definitions is the relation to the service concept, as also indicated in the name SOA. While the technical and architectural aspects behind SOA are mainly not new [KBS04], the alignment of business and IT along the service concept might be the distinguishing aspect of SOA. For this thesis, we use the following definition of SOA:

Definition 8

Service-Oriented Architecture

“SOA as a paradigm aims at an adequate support of business through IT-systems. Therefore, business and IT are conceptually aligned by the utilization of the service concept.” [ANT+11]

Similar to other paradigms, architectural styles, or design patterns, SOA is a way to capture and describe best practices for a certain type of systems (see also Figure 15). More characterization of this type of systems is given in the subsequent sections.

What SOA is NOT

Although SOA has a lot to do with business, it is not a paradigm tailored to a certain business domain like financials. Rather, each organization and business can build IT-systems according to their own needs along the principles of SOA. Consequently, SOA is nothing that someone can simply buy out of the box (similar to object-orientation as a paradigm for programming). SOA is not a product rather it has to be applied in a concrete context with appropriate engineering techniques.

Because SOA as a paradigm comes with so many aspects, it has often been misconceived. Several of these misconceptions will be presented in Section 2.2.2 and should support a better understanding of SOA. In Section 2.2.3, we present a conceptual model for SOA, which aims at a more in depth explanation of the different aspects of SOA, in particular Business, Architecture, Technology, and the role of Services and Engi-

neering. For this thesis, the focus is on architecture (which of course also involves business and technology aspects). An introduction into the architectural background of SOA is given in Section 2.2.4 and more details can be found in Chapter 6.

SOA or
NOT SOA?

One approach often helps to characterize a term: We give criteria for the decision whether a certain system is built according to SOA principles or not. Therefore, we give the following criteria which have to be fulfilled for a system to be recognized as a SOA system:

- Organization of the business to be supported by the IT-system according to service concepts (clear modeling of services and business process, e.g. as described in Section 2.2.3)
- Fulfillment of key architectural characteristics as described in Section 2.2.4 (self-contained services with clear interfaces, etc.)
- Alignment (clear mapping / tracing) between the services at business level and the services in the IT-system

2.2.2 SOA in Practice

SOA's
history

In early years, business was mainly supported by IT with very specialized and local systems which were often even not directly accessible for users. Later on, in the 1970s and 1980s, users got direct access to the systems with the emergence of terminals and personal computers. Then, a stronger integration of business functions into integrated product suites could be observed, as for example in SAP for ERP and financials. In the 1990s, even more integration evolved. Technically, standards and products emerged for distributed computing, like RPC, CORBA, or DCOM. All types of systems became integrated, first locally, then more and more globally, and also across enterprise borders, with the appearance of the internet. Integrations were often done on a one-to-one basis with considerable effort. EAI (Enterprise Application Integration) [Lin00] was one earlier concept addressing the integration issues.

The term SOA was introduced in 1996 by Gartner [SN96]. In the earlier days, SOA mainly aimed at the integration of systems with clearly defined services. With the appearance of Web Services [KBS04] a standardized technology was available, which promised more interoperability by open, standardized, XML-based protocols like SOAP, WSDL, and UDDI that were defined and maintained by W3C [W3C]. Many of the underlying concepts like interface definition languages had already been available in CORBA [KBS04]. The rather new idea was to see services as a contract between a service provider and a service consumer who are often not located in the same organization.

In the 1990s and 2000s there was more and more tendency to support users in their workflows or business processes. This required the interplay

of different systems in the workflow of a user and the orchestration of activities among many users by a workflow system. As business processes are a key differentiation factor for many companies today, there is the need for high customizability. This led to the evolution of the SOA paradigm that also business processes and their contribution to business services became part of the representation. Consequently, also the technologies evolved towards larger stacks of protocols, covering also execution of service orchestrations with languages like BPEL [BPEL]. The resulting technologies are large modeling frameworks and runtime solutions, often centered around a central communication infrastructure, the so-called Enterprise Service Bus (ESB) [Cha04]. ESBs are often based on Web Service technologies but add other infrastructure functionalities like managing services, transforming between different data formats, etc.

SOA is
wide-spread

In the meantime, SOA is quite wide-spread in industry [MER10]. Nevertheless, it is difficult to get reliable numbers about the real usage of SOA and about how much of today's IT is organized according to SOA. This missing transparency is also caused by the fact that SOA is understood and lived as strongly diverging ideas in different enterprises [MER10]. The range is from only using Web Services technology to provide easier means for integration with external systems up to the organization of a complete enterprise and its IT according to service-oriented principles. The case that mainly technologies like Web Services are adopted is quite common as our practical experience has shown.

SOA is
big business

Because many large enterprises see SOA as a promising paradigm to improve their IT's support for more productive business and to reduce the cost for IT itself, there is a big business around SOA. Particularly, technology vendors (selling so-called SOA-stacks, i.e. infrastructure components) and consulting services companies (often coming together in the same company) support enterprises to transform their IT towards SOA [GBD08, AH06]. Besides the enterprises using large IT systems, also many companies developing software products organize them along SOA principles for better integration into the customer's IT environment. The alignment of SOA with other trends like BPM (Business Process Management), EAM (Enterprise Architecture Management), or BAM (Business Activity Monitoring) seems to be a further source for many consulting projects.

SOA and
misconcep-
tions

Despite the wide adoption of SOA, many practitioners have been disappointed with their SOA introduction projects in the past. However, scientific or industrial publications rather focus on success stories. In internet articles and blogs there is a more open discussion on this topic going on (e.g. [Man09]). One key observation often described is that SOA is reduced to technology discussions about ESB or Web Services whereas the big picture, in particular the business aspects, are often neglected.

A key reason for disappointment of practitioners is that they did not achieve their goals with SOA projects. Typically, SOA is introduced in en-

vironments of highly complex, heterogeneous, distributed systems, with a focus on integration of legacy and standard software and the achievement of critical quality requirements like performance, scalability, flexibility, and security. These are a lot of challenges, which have caused a lot of problems for IT in the past and the expectation was that SOA would overcome these problems. Such expectations are not least caused by marketing claims of technology vendors and consulting companies.

Several expectations, which did not fulfill in practice, can be summarized as **common misconceptions** and have been widely observed in practice [LMS+07, NP07, Naa08, Pro11]. In the following, some of the common misconceptions are briefly described:

- **“SOA defines a system’s complete architecture”**: SOA defines rather an architectural style or a reference architecture than the complete architecture of a system. There are still many architectural decisions open, in particular how to represent the business logic in terms of a service-based architecture. Further decisions are necessary for constructing user interfaces, defining appropriate deployments, or addressing quality attributes like performance.
- **“SOA vendor stacks provide a system’s architecture”**: SOA vendor stacks are big collections of all types of technologies in SOA, like communication infrastructures as ESB, engines for service orchestration with BPEL, etc. Although such vendor stacks realize several key architectural decisions and solutions of SOA, they do not cover all necessary architectural decisions, as described also in the previous point.
- **“SOA leads to a high degree of flexibility and reuse”**: Only the fact that SOA comes with several architecture mechanisms that allow constructing for flexibility and reuse does not automatically lead to the achievement of these goals, as described and motivated in Chapter 1 of this thesis. Rather, a lot of further reasoning and engineering is necessary to get the architecture right.
- **“SOA enables interoperability by standardization”**: Interoperability of systems has many aspects, e.g. the syntactic interoperability and the semantic interoperability. By standardized communication protocols and the possibility to execute transformations SOA provides a means to overcome syntactic heterogeneity. However, there are a lot of further assumptions made in systems which often make practical integration a real challenge, even with a SOA.

Analyzing different misconceptions may lead us to the conclusion that SOA often offers the appropriate architecture mechanisms to achieve required properties a system. However, for the further decisions needed, sound engineering and governance is necessary, because they cannot be provided by architecture and technology only. As a conclusion, engineering is important also in the world of SOA, which is known in recent days

as Service-Oriented Engineering (SOE). The next section describes a conceptual model as a background for SOE.

2.2.3 A Conceptual Model for Service-Oriented Engineering

As described in previous sections, there is not THE definition for SOA which would be generally accepted or applicable. Further, we discovered, that sound Engineering might be one of the key missing aspects why many SOA initiatives and projects fail in practice. In order to have an adequate foundation for engineering methods, we decided to define a conceptual model that should put all the concepts and aspects of SOA into relationship, which is not possible in a crisp definition. This model is described in [ANT10, ANT+11] and will be briefly introduced in this section as context information for this thesis.

Goals and inspiration

The key goals for the definition of the conceptual model are the following:

- **Unification of perspectives:** As described in Section 2.2.1, many different perspectives on SOA exist, of which most are somehow right, but seem to be incompatible. Our conceptual model aims at covering all these aspects at least in a way that an easy mapping of the known perspectives is possible in order to allow discussions focused on content rather than on terminology.
- **Foundation for communication with customers:** All enterprises and people have some perception of SOA, which might be restricted to one specific perspective. Talking about SOA with customers requires to sketch the big picture and to be able to clearly describe the role of the customer in the overall SOA model. This needs to be covered in the conceptual model.
- **Foundation for service-oriented engineering and methods:** Engineering IT systems based on service-oriented concepts requires a thorough understanding of the relationships between business aspects and potential solutions in IT. The conceptual model has to express all these aspects in a consistent and traceable way as a basis for derived artifact models and process models.

There is also earlier and related work detailing the SOA paradigm with conceptual models. The standardization organization OASIS published two documents called reference model [OAS06] and reference architecture [OAS09] which describe models with an intention similar to ours. The SOA method described by sd&m in [EHH08] comes with a conceptual model, too. Our model is not intended to come up with something completely new. Rather, it is strongly inspired by the two sources named and it adds the aspects important to us. These are in particular the following aspects:

- Seamless coverage from business concepts to IT systems
- Strong focus on engineering, in particular on the quality of the systems considered
- Strong focus on user perspective in addition to enterprise perspective
- Explicit description of all types of organizations involved around SOA (usage, operation, development)

System type
in focus

SOA as a paradigm is not a general-purpose development paradigm for building any type of IT-systems. Rather, it is dedicated to a certain type of systems with common characteristics and shared challenges. Only by such a level of specialization of system types [GAO09], the solution aspects in terms of architectures (see also Section 2.1.3), technologies and tailored engineering methods can be specific. In the following, some common characteristics of typical SOA-based systems are listed (taken from [ANT+11]).

- supporting of business and business processes
- serving multiple users
- constituting system landscapes (systems-of-systems)
- integrating multiple heterogeneous systems
- integrating legacy or COTS systems
- being complex and distributed
- being developed for long life-time and maintenance
- being continuously evolved following business
- being controlled by distributed responsibilities

Key ideas

Our conceptual model is based on the following key ideas which are here briefly described below:

- **View-based documentation** and **glossary**: The number of elements described in the model is too large to be described in a single diagram. Therefore, we decided to represent it in different views showing coherent aspects.
- Organization in **5 key areas** (see Figure 16): *Business, Architecture, Technology, Service, Engineering*. **Business** expresses the real world which is to be supported by the IT-systems at hand. **Technology** is the technological world of a physical realization and operation of IT-systems. **Architecture** is the set of principal design decisions made in order to achieve IT-systems which fit the business' needs. **Service** is included as a key area as it is seen as the linking theme in service-orientation which spans from business over architecture to technolo-

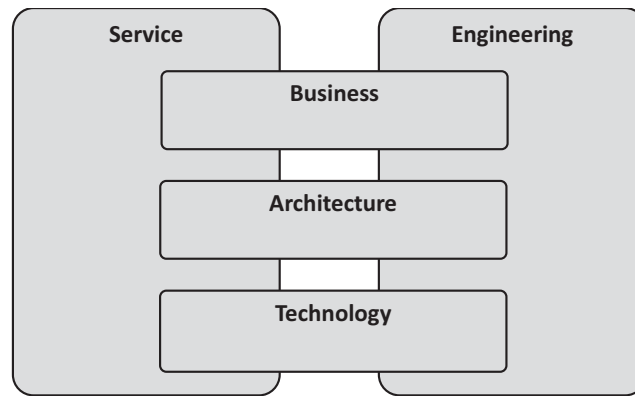


Figure 16: Key areas of service-orientation

gy. Finally, **Engineering** is included as an area which is needed to construct the IT-systems at hand in alignment with business. The organization in these areas is also inspired by [Mas07], where the areas Architecture, Enterprise, Computing, Platform, and Engineering are distinguished.

- We distinguish three main classes of services which are necessary to express all constellations of enterprises and the business services they offer to each other. The first, and typically anticipated, category is called **Plain Business Service** in our model. This can be any service, organizations offer or consume which might be totally independent of software or supported with software. An example would be a logistics service. The other two service types became necessary to bring more light into very general sentences like “company X offers a service for stock exchange values”. This might be a company offering the plain business service delivering data. Or it might be also a company operating such a service. Alternatively, it might even be a company developing the software for such a service. Or a combination of the aforementioned. Thus, we introduced two new service types: **Operation Service** and **Software and Systems Engineering Service**. First this classification is only at the business level but has of course several connections to services at the software level.

Views and glossary

As described above, the conceptual model is described with the help of views, showing particular aspects of the model. In total, there are currently 14 views, each described with the rationale behind the view. Additionally, there is a glossary explaining all the elements in the conceptual model. In order to give an idea what such views look like, Figure 17 gives some insight. It describes with a focus on the service term which types of services we distinguish in our model, and particularly how they relate to each other. All details can be found in [ANT+11].

Architecture as the core topic of this thesis has a specific role in service-orientation, too. This is also part of the conceptual model, mainly what it

means to look at SOA as an architectural style or a reference architecture. These aspects are described in more detail in the following section.

2.2.4 SOA as Architectural Style and Reference Architecture

SOA is often perceived as defining an architectural style or a reference architecture as a blueprint for designing software systems. Because of the architectural focus of this thesis, we describe these aspects of SOA as a foundation here. More details, in particular on the support of SOA for flexibility, are presented in Chapter 6.

Architectural styles and reference architectures

Both, architectural styles and reference architectures aim at a common goal, but with a different focus. They provide proven solutions and best practices for recurring challenges in the construction of IT systems. By focusing on a class of systems with more or less abstract commonalities, for example the need to achieve particular quality attributes like maintainability, it becomes possible to find reusable solution concepts. Besides describing an abstract solution idea, architectural styles and reference architectures also provide a common language that eases communication among stakeholders in software development.

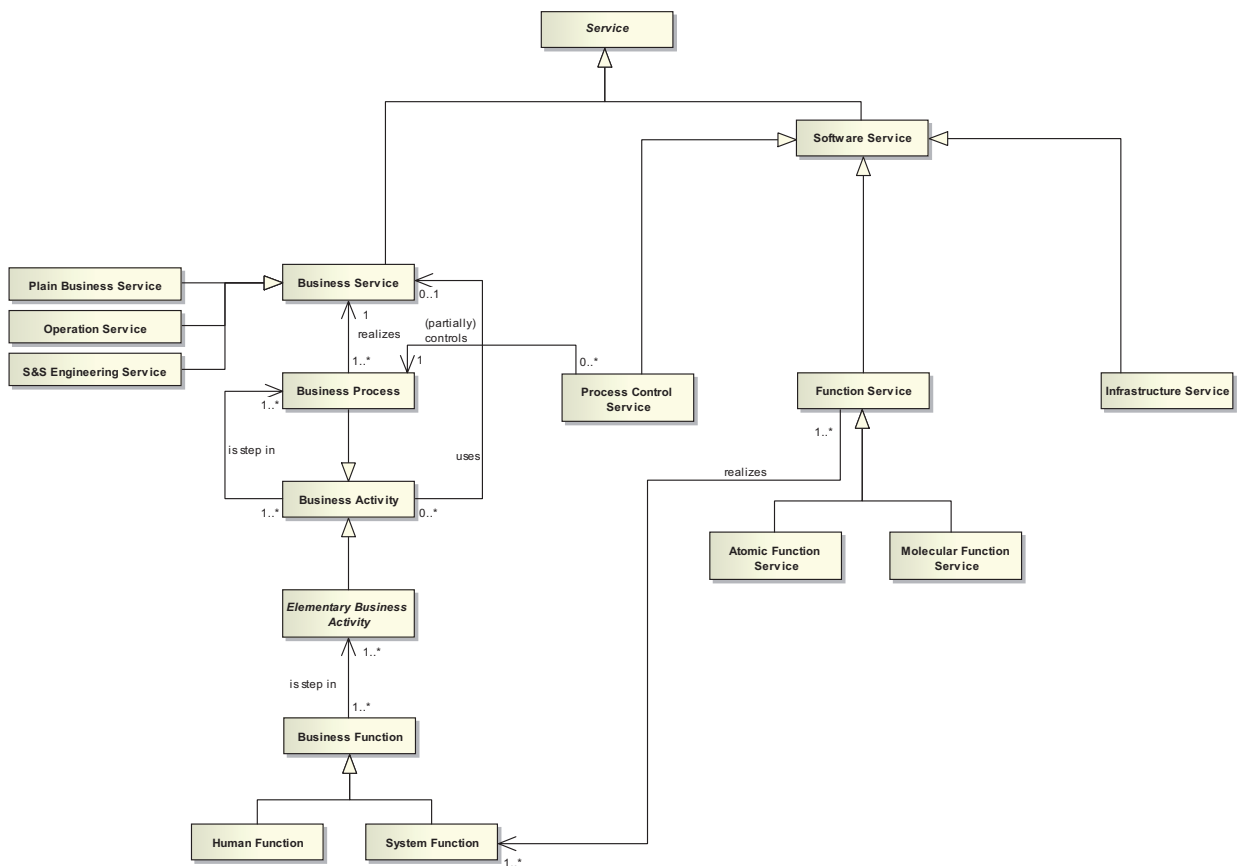


Figure 17: Example view from conceptual model for service-orientation

Architectural styles [GS94, BCK03] define component types, connector types among these components and rules how a system might be constructed from these components and connectors. Architectural styles typically do not have any prescription about functionality of a system, rather they define only abstract meta-elements that can be used to organize the functionality and allow communication among such elements. Well-known examples of architectural styles are the layering style, black-board style, or client-server style.

Reference architectures [NAB11] define more detailed but still abstract architectures that are shared by a class of systems in order to solve their common challenges. Reference architectures can also be domain-specific and thus might give a standard decomposition of the domain-specific functionality. Reference architectures are often composed of multiple architectural styles and typically cover the whole scope of a system, whereas architectural styles only cover certain aspects quite abstractly. The level of detail of reference architectures can vary widely due to different purposes of reference architectures. A well-known example is the reference architecture for Java Enterprise Applications [JEE].

Challenges addressed in SOA

As described in Section 2.2.3, SOA also aims at providing solutions for common challenges in the targeted system class. Such challenges are at an abstract level:

- Flexibility for changes following an evolving business
- Distribution serving multiple users and their business processes
- Integration of heterogeneous systems from multiple sources

SOA as architectural style

The so-called “SOA-Triangle” [Erl06] (see also Figure 18a) is widely perceived as the architectural style prescribed by SOA. In particular due to the spreading of Web Services [KBS04] as a realization technology, it has

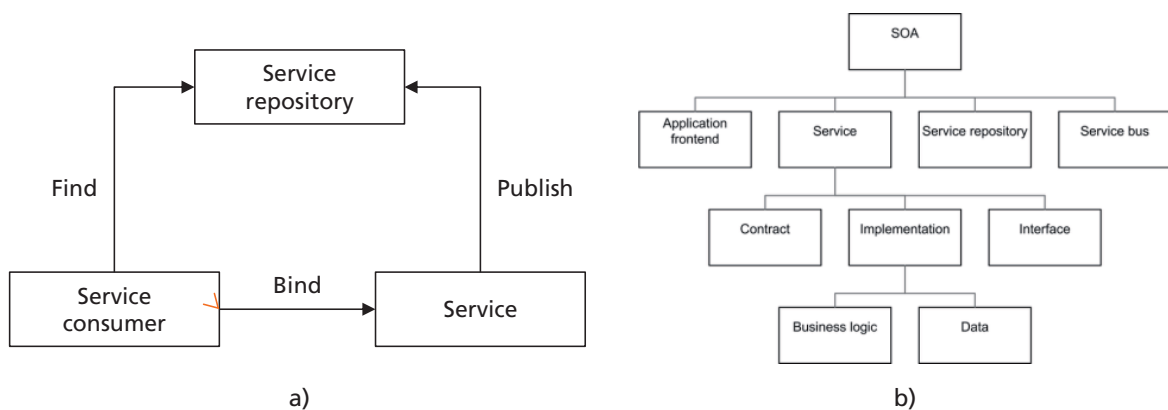


Figure 18: a) SOA triangle [Erl06] b) SOA element types [KBS04]

gained popularity. It prescribes that there are service providers and service consumers. The key connection is that a service consumer requests a service provided by a service provider. This can also be done indirectly by querying a service registry which returns at runtime an address of the service provider and thus facilitates a kind of runtime flexibility in selecting an appropriate service provider. It is important to note that this type of flexibility is not the one in the focus of this thesis which deals with changing the system at development time (see Section 1.5). A similar definition of the architectural style defined by SOA is given in [KBS04] (see also Figure 18b): There, it is made more concrete that also application frontends are needed which offer the user interface and consume services, whereas the main business logic is offered via services. The service registry is also there; in addition a service bus is added which is a more concrete representation of the connector responsible for the communication among service providers and service consumers.

Beyond this initial description of element types, there are also proposals on the properties of single services [e.g. Erl06, KBS04, HHV06, Jos07]. It is proposed that services should be stateless, idempotent, technology-agnostic, etc.

SOA as reference architecture

Based on the SOA architectural style, several reference architectures are currently available. The most prominent example is the IBM S3 (Service-oriented Solution Stack) SOA Reference Architecture [AZE+07a, AZE+07b]. This reference architecture is a compilation of best practices in IBM’s SOA consulting projects and comprises several architectural aspects. They are covered in two views (see Figure 19): The solution stack view and the middleware view. These two views cover important design ideas like the separation of business processes and services, or different types of services and their communication via a service bus. This reference architecture does not cover any domain-logic related parts, but it offers the placeholder elements how to organize concrete business logic.

Summarizing, the term SOA is used for an architectural style, a reference architecture and finally also for the concrete architecture of a system or

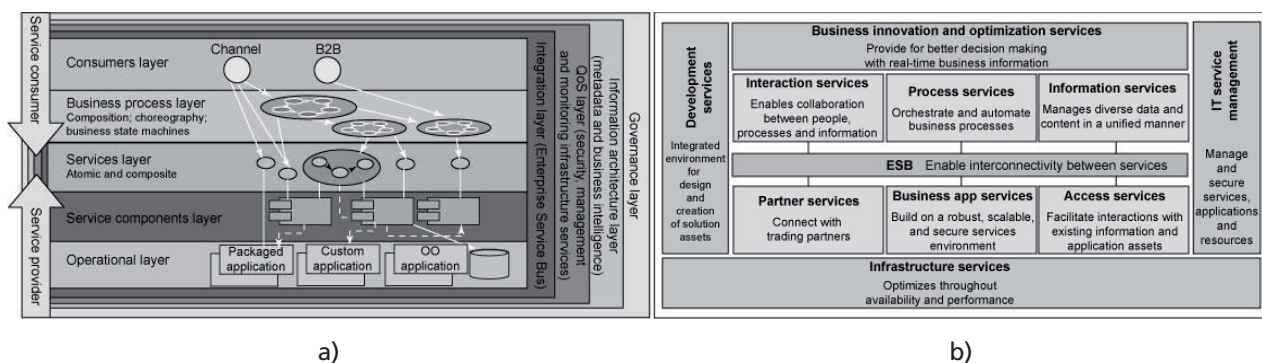


Figure 19: a) Solution stack view b) Middleware view [AZE+07a, AZE+07b]

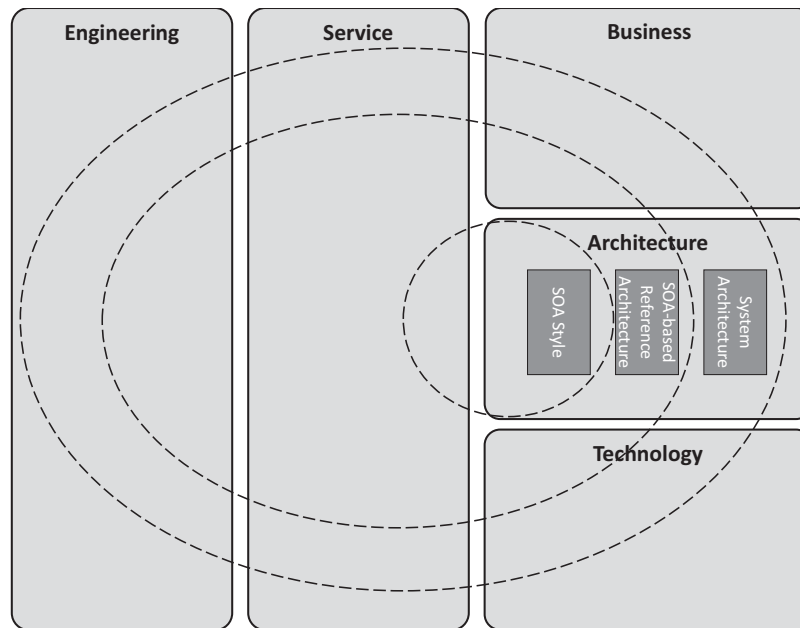


Figure 20: Facets of SOA and their relationships

system landscape following the ideas of SOA. The relationship between these aspects is depicted in Figure 20. A SOA-based reference architecture is typically based on the SOA architectural style. A concrete SOA-based system architecture is typically based on a reference architecture or at least on the SOA architectural style. This categorization also has a close relationship to the five key areas of our conceptual model we introduced in Section 2.2.3. By means of dashed ellipses Figure 20 shows abstract areas which denote a different coverage of the five key areas and the decisions taken about software in these areas. While the architectural style only covers some architectural concepts and the service concept, the reference architecture already incorporates aspects of business and technology. The concrete system architecture has the largest coverage as it has to resolve decisions that might be left open by architectural styles or reference architectures. For example, a concrete system architecture has to care about all concrete business-specific instances of business processes and services.

Technologies for SOA

In Section 2.1.3, Figure 15 described how architecture domain competence and best practices can be captured (challenges, solutions, technologies). So far, we describe key challenges and solution concepts of SOA-based systems. For SOA, many technologies in form of frameworks, platforms, protocols, etc. exist [Jos07, KBS04]. In the following, we will list some of the most prominent ones:

- Web Services
 - Service-Interface-Language (like WSDL)
 - Communication languages (like SOAP)
- Enterprise Service Bus (ESB)

- Process Execution Engines (with or without generation of graphical user interfaces) (like for BPEL)
- REST (Representational State Transfer)

2.2.5 New Paradigms in Service-Orientation

Cloud Computing Today, the term “service” is prominently found in another context: the different service models [MG09] in cloud computing are called:

- Software-as-a-Service (SaaS)
- Platform-as-a-Service (PaaS)
- Infrastructure-as-a-Service (IaaS)

The key idea behind these services is that the consumer of the services can use software (SaaS), runtime execution platforms (PaaS), or infrastructure entities like computational power or data storage volume (IaaS), which are provided and particularly operated by the service provider. By offering such services to multiple customers, the service provider can achieve and realize economic benefits (economies of scale by sharing resources, balancing resources between consumers, more efficiency in management of resources), which makes cloud computing a successful business model in today’s IT.

Although SOA and cloud computing share common ideas around services, there are significant differences. Whereas SOA mainly targets at the construction and integration of system landscapes, cloud computing targets at outsourced and scalable operation of IT services.

The quality attribute flexibility is also often mentioned in the context of cloud computing, but with a different meaning than the one underlying this thesis. The expectations for flexibility in cloud computing mainly target at changing and adapting providers, services, or resources [IBM11]; the change of the software itself is rather out-of-scope.

3 Flexibility: State of the Art

"The only constant is change"
Heraclitus

Flexibility as a quality attribute of software has many facets. This reaches from the principal understanding of what makes software flexible over constructive and analytical methods for engineering to concrete architectural mechanisms and technologies that can induce flexibility in a certain domain of systems. Consequently, also the analysis of state-of-art around flexibility has to consider these facets. This chapter presents the current state-of-the-art around flexibility and describes how it contributes to the research challenges and research ideas addressed in this thesis.

Approach

To contribute solution aspects to the practical problems described in Chapter 1, we derived research directions to which this thesis contributes. We also described in Chapter 1, which research challenges we derived for focusing and which research ideas are supposed to address the challenges. These research challenges and ideas are obviously depending on the current state-of-the-art in the sense that they build on many existing ideas and exceed them in certain points. While Section 1.4 only

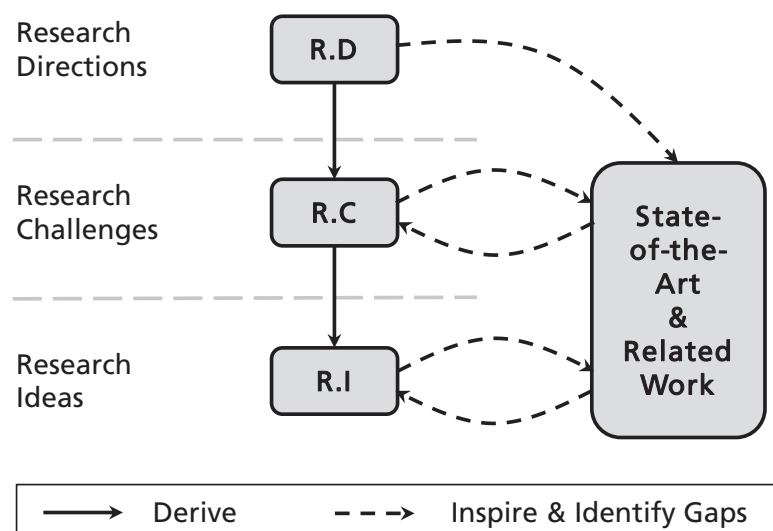


Figure 21: State-of-the-art in the context of research directions

outlined the challenges and ideas, this chapter presents the background on existing work.

Figure 21 describes the role of state-of-the-art and related work in the context of this thesis. From our research directions, the areas of related work are derived. The identification of research challenges and research ideas is an iterative process of shaping the contributions of this thesis by inspirations of state-of-the-art work and the identification of gaps in this work which would be worth to be filled.

Research Directions

The following research directions are approached in this thesis (see Section 1.4). They are used for an according categorization of related work (summarized in few words):

- **R.D1:** Theoretical foundation of flexibility and architecture [3.1]
- **R.D2:** Constructive support of flexibility [3.2]
- **R.D3:** Flexibility measurement and evaluation (automated) [3.3]
- **R.D4:** Flexibility for SOA (mechanisms and how to use them) [3.4]

The following sections describe related work belonging to the research directions. Obviously, research approaches are not always matching exactly one research direction, but we will assign them to the most appropriate one. Section 3.5 summarizes the related work and the gaps found and explains how our research challenges can be derived.

3.1 Flexibility as a Quality Attribute

In this section, we first describe flexibility as a quality attribute in a field of other similar quality attributes (3.1.1). Then, we describe flexibility as researched in the area of information systems (3.1.2) and in other disciplines like systems engineering for space ships (3.1.3). Finally, we compare flexibility to the key characteristic in product line engineering, which is variability (3.1.4).

3.1.1 Flexibility and Related Quality Attributes

Development time quality attributes

Flexibility is one among several so-called development time quality attributes. That is, these quality attributes denote how well a software system supports development activities. Many different terms and definitions for development time quality attributes are around, e.g. maintainability, evolvability, flexibility, changeability, modifiability, adaptability. Mostly, they are not precisely defined and interchangeably used in practice. One reason for difficulties in distinguishing these quality attributes is that their concrete meaning for a system can only be expressed with the help of architecture scenarios [CKK01], but not with a brief definition of

the quality attribute. Consequently, the definition of quality attributes can give only rough directions and characterize quality attributes with certain properties. In this section, we refer to typical definitions to show how flexibility can be delineated from other quality attributes. Figure 22 shows a sketch of refinement relationships among quality attributes; however, according to different definitions it could also look quite differently.

Maintainability

Maintainability serves as our starting point for exploring the quality attributes. Maintainability is probably the most frequently used name for quality attributes in the area discussed [IEEE90, ISO1926, Bar03]. Typically, maintainability is characterized via different types of changes (corrective, perfective, adaptive), and it covers several aspects that are needed to allow easy changes to a software system. First, there is of course the inherent property of the software to require only little portions of the software to be touched or not (changeability / modifiability). Second, there is the question how fast engineers can understand the system to conduct changes (analyzability). Third, there is the question how well the system can be tested after conducting the change (testability). A more detailed quality model for maintainability is described in [BDP06, DWP+07, WDF08], where maintainability is defined via a two-dimensional matrix of maintenance activities vs. artifacts impacted by these activities.

Modifiability
Flexibility

Modifiability [BCK03] is often seen as the property of the software system to handle changes locally. Therefore, architectural mechanisms are used [BCK03] in order to control the impact of certain changes. In the understanding of this thesis, flexibility is that part of modifiability which

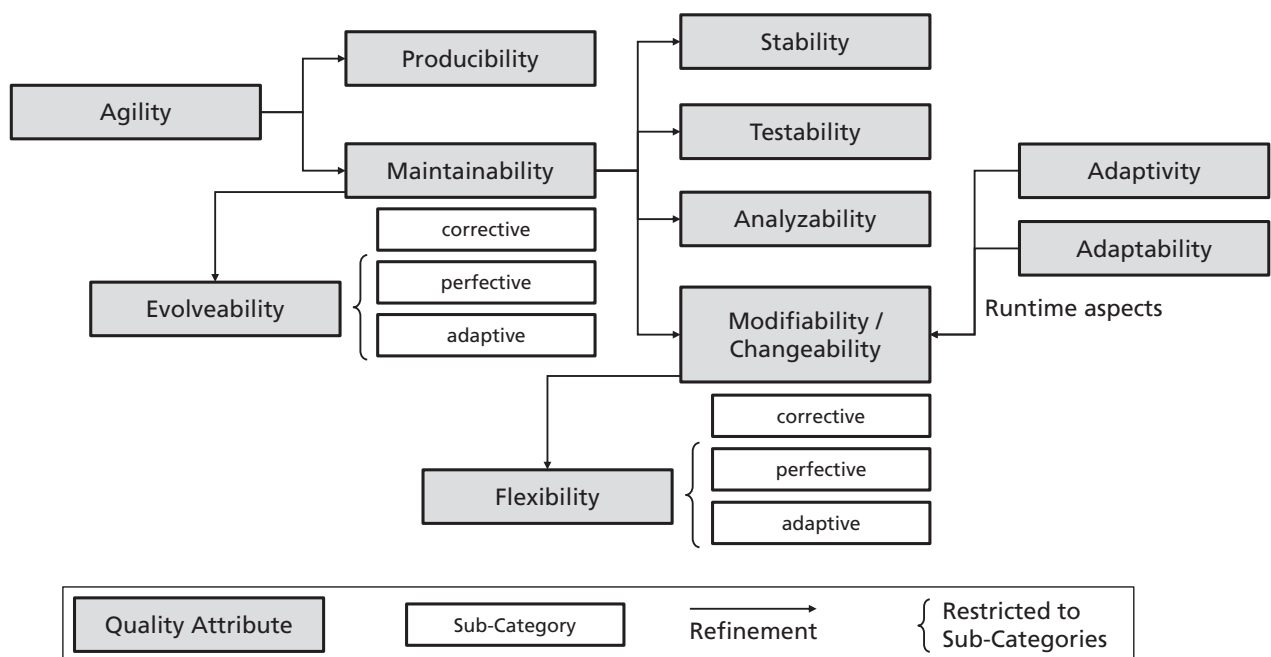


Figure 22: Relationships among quality attributes

has no corrective aspects. Other definitions of flexibility [BKL+95] have a broader focus.

Evolvability Evolvability is an often found term, too, which spans across many aspects of change [MM98, BCE08, BC10]. We classified it here as being everything concerning changes except for corrections of a system.

The ultimate goal of a company relying on software is Agility [Sch04], which means on the one hand that software systems have the properties to be easily changed (maintainability) and that on the other hand the organization is able to conduct these changes efficiently. Therefore, optimal processes and organizational prerequisites are necessary, as well as adequate alignment of processes and organization with the system's architecture (producibility [Car12]). Please note that this interpretation of Agility does not directly match to agile development processes.

Adaptability
Adaptivity Further aspects of change are often called adaptability or adaptivity [Che08]. However, these changes mostly mean that a system is prepared to adapt to different contexts during runtime. That is, all change behavior is already built in and has to be selected and activated.

For all characteristics and definitions given above, different terminologies and definitions exist. It is not the intention of this thesis to come up with consistent definitions of all terms. For the context of this thesis, a precise definition and scope of Flexibility is given in Chapter 4.

3.1.2 Flexibility in Information Systems Research

Information Systems are one type of software systems which are often described as opposed to Embedded Systems. In that sense, they are a particular system class or domain in which the methods and technologies of software engineering are applied. Beyond this definition, there is another interpretation of Information Systems: the one which mainly cares about the business to be supported by IT-systems and how this can be best done³. There is a lot of research on Information Systems around that is often rather dedicated to economic sciences.

From this perspective of Information Systems, flexibility is an important quality attribute of software systems and some research exists, mainly by Judith Gebauer, Franz Schober et al.

In [GL05], a conceptual model for flexibility in information systems is introduced. A major distinction is made between flexibility-to-use and flexibility-to-change. While flexibility-to-use is a property which is visible to the user of a system due to the possibility to provide different behavior

³ In German: Wirtschaftsinformatik

in different contexts, flexibility-to-change is the kind of flexibility which is also in the focus of this thesis.

In [GS06], an extension is provided which is based on the characterization of business processes. Three major characteristics are described: Uncertainty, Variability, and Time Criticality. In combination with decision variables that indicate whether to go for flexibility-to-use or for flexibility-to-change, calculation models are presented which allow determining business process performance in terms of cost. In [GL08], based on the elaborated model, strategies and considerations for the introduction of new information systems in enterprises are presented. The latest publication [SG11] is based on the calculation models and is dedicated to the question of determining the value of flexibility for information systems. Whereas these publications deal with detailed mathematical models concerning the economic value of flexibility, they do not go into detail about achieving flexibility as it is in the focus of this thesis.

Business Process Management (BPM) is a discipline that unifies parts of the economic world and of the technical world (software systems support business). There is a strong need for flexibility in business processes and thus in the supporting IT systems as well [OS03, ENS07]. Business Process Management Systems often explicitly target at process flexibility by decoupling processes from functions and making processes descriptive first class development artifacts (see Chapter 6).

3.1.3 Flexibility in other Disciplines

Flexibility as a quality attribute is not only important for software systems, but also for other types of systems or products. Consequently, there is research on flexibility in other disciplines, which is briefly sketched in this section. Most of the research is conducted in the environment of Massachusetts Institute of Technology (MIT).

Industrial products

In the area of industrial goods and products, for example electronic tools, there are publications about methods for the analysis of flexibility. In [RWC+03], a method called CMEA (change modes and effects analysis) is described. It works by decomposing the product into modules and parts and then reasoning about potential change causes, potential changes on the causes, and potential effects of the changes. Then, the flexibility is ranked with the help of a table which assigns a value of 1 to 10 (10 is lowest impact) to the estimated effects. Additionally, the occurrences of changes and the readiness for changes are rated in a similar way. Finally, the overall flexibility is calculated taking all potential changes into account. In [RWC+05], further case studies of applying the method are presented. The method is purely analytical and based on an idea similar to change scenarios. Further research in this direction is presented in [TSW09], but with a different way of measuring: There, the so-called

high-definition design structure matrix is used for the representation of change impacts and interdependencies.

Space science

More research on flexibility can be found in the domain of space vehicles like satellites. Such systems often have a very long life-span and are no longer physically accessible having been launched. Nevertheless, changes are necessary to react to recognized situations, which means they need more flexibility. Early publications [SHN01, SHN03] start with the observation that flexibility is important for many disciplines but is not well-defined and often mixed up with other quality attributes like robustness. One reason is the valid but too simple definition “Flexibility is the ability to handle change”. In order to further explore flexibility, the authors pose some questions which are a good foundation for any type of quality attribute to explore, and which are also tackled in our thesis:

- “What is flexibility? How does a formal definition look like?”
- “Why or when is flexibility needed in system design?”
- “How can one design for flexibility? What are the design principles?”
- “What are tradeoffs associated with flexibility?”

The papers [SHN01, SHN03] focus on the first question and therefore broaden the definition of flexibility with the following aspects, which are also included in our characterization: 1) time and occurrence of change during system life-cycle; 2) characterization of what is changing; 3) clear metrics for flexibility. Their resulting definition is: “Flexibility is the property of a system that allows it to respond to changes in its initial capabilities and attributes – occurring after the system has been fielded, i.e. is in operation, in a timely and cost-effective way”. In a later publication [RRH08], the definition is further refined, mainly with the focus of describing changeability as the core concept of other quality attributes (modifiability, flexibility, scalability, adaptability), which is mainly in line with our classification in Section 3.1.1. They add three aspects to classify change: 1) change agents: where does the change originate (external or internal to the system); 2) change effects: what changes in the system or is kept constant on external triggers; 3) change mechanisms: what is necessary to bring the system from the original state to the target state of change. Further work on calculation of flexibility metrics is published in [NHJ05, SWV+08]. Additionally, several doctoral theses have been published in this research area: 1) [Sal02] describes how to weave time into system architectures as an enabler for flexibility; 2) [Suh05] describes the design of flexible product platforms; 3) [Nil05] describes a framework concerned with the value of space system flexibility.

Buildings

Building houses needs dealing with flexibility, too. [TMD09] describes an analogy to software with typical aspects of a house that are easy to change (e.g. the furniture) or hard to change (the main walls). The different levels of ease of change are described as “Shearing Layers”.

3.1.4 Variability in Product Line Engineering

Product Line Engineering (PLE) is a discipline in software engineering which aims at generating benefits from the fact that multiple but similar systems have to be built. The idea is to realize savings by explicitly exploiting the commonalities among different systems and by managing their variabilities [CN07, LSR07]. Variability [GBS01] is one key property in PLE which is challenging to handle for development organizations. Managing variability means to minimize the impact of variations and to localize variations in well-defined variation points. In that sense, variability is similar to flexibility. During design of software product lines, the elicitation of commonalities and variabilities is a key activity. Often, this activity is called Scoping [Sch02] and aims at coming up with economically feasible product instances and well-defined commonalities and variabilities. In that sense, scoping has similarities to the elicitation of flexibility requirements. Further, product lines evolve over time like single systems do. Thus, there is also the need for flexibility in product lines. Elicitation of evolution requirements is supported by methods as described in [VDG08, VEG08].

3.2 Construction for Flexibility

Flexibility as a quality attribute is mainly addressed analytically in research. That means, there are dedicated methods and metrics as to how to evaluate the flexibility of a certain software system. This research is explored in Section 3.3. Constructively, there is not much support for flexibility. In this section, we summarize how contemporary architecture methods support definition of flexible architectures. First, we give an overview on the elicitation of flexibility requirements (3.2.1). Second, we look at architecture definition methods (3.2.2). Third, we describe architectural mechanisms for flexibility (3.2.3) which can be applied during in the process of designing. Finally, we present how SOA design approaches support flexibility, a key quality attribute in SOA (3.2.4).

3.2.1 Elicitation of Flexibility Requirements

Flexibility is a quality attribute or also called non-functional requirement (NFR). In the field of requirements engineering, many methods exist (e.g. TORE, Task-Oriented Requirements Engineering [PK04, ADE+09]), which support the elicitation of functional as well as non-functional requirements. Additionally, there are more specialized methods to elicit quality attributes and their particular meaning for a software system at hand (e.g. [Doe11]). Such methods typically support the requirements engineer with a systematic approach and with guidelines characterizing the quality attributes [VEG08, VDG08]. Flexibility as a development time

quality attribute is often not as well supported as other quality attributes that are directly visible for the user of the system.

As an input for architecture design, it is a wide-spread practice to express quality attributes precisely with architecture scenarios [BCK03]. In particular the flexibility evaluation methods as described in Section 3.3 heavily rely on architecture scenarios [e.g. BB99, BB00, BB01, Ben02, LRV99a, LRV99b].

Stating flexibility requirements is in a way a prediction of what will be needed in the future. Therefore, flexibility requirements are often uncertain. That is, it is not clear whether the requirements stated will ever be realized and it is not clear whether all important changes have been foreseen. Lassing et al published a study on how well flexibility requirements have been foreseen in a specific context [LRV99b, LRV03]. Bengtsson and Bosch conducted an experiment on identifying change scenarios [BB00] and found that groups come up with better scenarios than individuals. Interesting observations of the experiment are that nearly always changes to a database and operating system are assumed changes and that change scenarios cover significantly more often changes to interfaces and hardware than to the application logic.

3.2.2 Architecture Definition Approaches

In this section, we explore architecture definition approaches for their support for the specific quality attribute flexibility.

Very early work in the direction of flexibility has been done by David Parnas. In [Par72], he writes about decomposition of systems into modules, which is a very early approach for architecting software systems. There, he already brings the idea in to determine likely changes (similar to change scenarios) and to encapsulate the changes. He proposes to follow the principles of *Information Hiding* and *Localization of Change*. In [Par79], this work is followed up. In [Par94], Parnas describes the phenomenon of software aging and demands planning for change (“To apply this principle [design for change], one begins by trying to characterize the changes that are likely to occur over the “lifetime” of the product. [...] Since we cannot predict the actual changes, the predictions will be about classes of changes.”)

In our problem statement (Section 1.3), we identified the need for alignment of architectural mechanisms and business logic mapping in order to achieve flexibility. Contemporary architecture definition approaches can be classified in three major groups, according to their support for architecture mechanisms and to the decomposition of business logic.

Focus on quality attributes	First, there are approaches mainly concentrating on the achievement of quality attributes, like [BCK03] by Bass, Clements, and Kazman. The focus is on the design for quality attributes, which is also reflected in the name of the design method: <i>Attribute Driven Design</i> . Implicitly, there is of course also the assumption that the system under design is first functionally decomposed and then the mechanisms are applied, but there is little guidance on how this decomposition is done and in particular on how it is aligned with the architecture mechanisms to achieve quality attributes like flexibility.
Focus on functional decomposition	Second, there are approaches mainly concentrating on the decomposition of the domain and on the system in components of a software architecture. [ABB+02] for example completely neglects the design for quality attributes in the design process and focuses only on functional decomposition. Further approaches in this category are the SOA modeling approaches as described in more detail in Section 3.2.4. Siedersleben describes in [Sie04] an architecture approach which focuses on functional decomposition, but with a strong focus on the separation of business logic and infrastructure. Thereby, he does not focus on system-specific quality attributes but rather presents typical reference solutions which can be entitled as best practices and might work for a larger amount of systems.
Approaches covering both aspects	Third, there are approaches that incorporate aspects of both, functional decomposition and design for quality attributes. Bosch describes in [Bos00] an approach that iteratively decomposes a system and then applies so-called <i>Architecture Transformations</i> to achieve quality attributes, which means to introduce architectural mechanisms like styles. Several further architecture approaches exist that cover both aspects, e.g. [Kru03, Gor06, TMD09, Fai10]. It is common to all these approaches that they address the interplay of architecture mechanisms and the resulting elements of a functional decomposition only very roughly. That is, an alignment of architectural mechanisms and business logic mapping as needed for flexibility in particular, is not part of these methods. This is a gap that is identified as an open research question in this thesis.

Besides the methodical aspects, the knowledge about architectural mechanisms to achieve flexibility is crucial. The next section explores the state-of-art concerning such mechanisms.

3.2.3 Architecture Mechanisms for Flexibility

Architecture mechanisms	In Definition 4, the term Architecture Mechanism is defined. We use the term architecture mechanism as there are many different terms (style, pattern, tactic, etc.) in literature which have similar meanings but are also used differently in some contexts. Architecture mechanisms are applied to solve certain requirements, in particular quality attributes which cannot be achieved by a simple decomposition and which often affect
-------------------------	--

many components of a system. For example, flexibility or performance require the usage of architecture mechanisms. Architecture mechanisms are often described as best practice solutions to recurring problems in a certain context, often known as patterns.

Principles	Architecture mechanisms for flexibility typically base on abstract and general software engineering principles like separation of concerns [Dij82, ER03], information hiding [Par72, Par79, ER03], low coupling & high cohesion [SMC74, ER03] and make them more concrete.
Modifiability tactics	In [BCK03, BBN07], the term <i>Modifiability Tactics</i> is used. The key goal for modifiability is localization of change and avoidance of ripple effects (see [Bla01]). Further work in this context [OKK07a, OKK07b] deals with the value achieved by introducing architectural tactics and patterns.
Styles supporting adaptation	In [TMD09], architectural adaptation is motivated and described. There, the term adaptation is also mainly in line with our flexibility definition. Architectural styles are mentioned as the mechanisms supporting change. For example, the styles Application Programming Interface (API), Scripting Languages, Plug-Ins, or Event Interfaces are mentioned.
GoF patterns & POSA	There are several books available specializing on patterns for design and architecture. The so-called Gang-of-Four (GoF) patterns [GHJ94] are originally used at a more concrete design level, but many of the ideas can be applied at the architectural level, too. Often, these patterns target at the separation of concerns and thus they can contribute to flexibility if appropriately used ("Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change." [GHJ94]). A further source of many patterns is the Pattern-Oriented Software Architecture (POSA) [BMR97] series, which also provides many patterns supporting flexibility.
Evolution styles	Evolution styles [GBS+09, GS09] are no typical architectural styles, but the idea is quite close to supporting an architect in conducting changes to a software system. An evolution style captures a domain-specific set of evolution paths which determine constraints on how evolution has to take place starting from an initial architecture and resulting in a target architecture.

3.2.4 Design Approaches for SOA

Due to the popularity of SOA, several design approaches specializing on the properties of SOA-based systems were defined. SOA design approaches mainly focus on functional decomposition of systems and the mapping of functionality on the architectural element type *Service*. That is, these approaches rather focus on the properties of single services than on the overall architecture. Thus, design for flexibility is mostly no inherent property of these methods.

sd&m	Quasar Enterprise by sd&m [EHH08] is an extensive approach for designing service-oriented application landscapes which gives detailed guidelines on the step-wise analysis and design of SOA-based systems. Additionally, typical challenges like integration and the usage of SOA technologies are addressed. In [HHV06], more information on desirable properties of single services is described, like coarse-grained or context free.
IBM	IBM specialized the Rational Unified Process (RUP) [Kru03] towards a method for service-oriented systems: It is called Service-Oriented Modeling and Architecture (SOMA) [AGA+08, AA06]. SOMA describes how to analyze the business to be supported and then how to identify and refine services which adequately support the business and its processes. Zimmermann developed a further approach in IBM, which is described in [Zim09]: A framework is developed which supports making and modeling architectural decisions for service-oriented systems. The focus in this work is on making knowledge about SOA-design explicit and use it to guide architects in designing their own systems.
Erl	Erl describes a method for service-oriented analysis and design [Erl06]. He decomposes the business and its business processes and identifies service candidates which are assigned to three categories (entity services, task services, and utility services). Erl gives extensive guidelines on the decompositions, but focuses on the functionality only.
IESE	Lee et al describe an approach for developing service-oriented product lines [LMN08, LMN10]. The analysis step is based on feature models and the features are translated to services. Therefore, guidance on service identification and definition is given.
SEI	The Software Engineering Institute (SEI) describes in the report "Architecting Service-Oriented Systems" [BLM+11] typical properties and challenges of service-oriented systems. Less focus is on the method. Rather, the architecture mechanisms like Enterprise Service Bus or Business Process Engine are discussed in the light of their impact on quality attributes.
OMG	The Object Management Group (OMG) defined a new UML-based language standard for the description of service-oriented systems [OMG09]. The language extensions are mainly meta-model elements and UML stereotypes, adapted to the SOA terminology.
Gebhart	Gebhart proposes an extension for SOA design approaches which focuses on a high-quality design [Geb11]. As quality, he proposes properties like <i>Unambiguous Categorization</i> , <i>Retrievability</i> , <i>Loose Coupling</i> , and <i>Autonomy</i> for individual services. For these properties, he introduces metrics and proposes how to identify problems regarding these properties and how to improve the design.

All approaches outlined focus on the definition of single services and lack support for proactively designing quality attributes like flexibility into a system.

3.3 Measurement and Evaluation of Flexibility

Measuring and evaluation of flexibility is in the focus of several methods and approaches. Different names for the quality attribute under evaluation are used, mainly maintainability, modifiability, and flexibility, but the general ideas are mostly transferrable. In the following sections, we will briefly sketch different research directions and approaches for the evaluation of flexibility.

3.3.1 General Overview on Architecture Evaluation Methods

Architecture is the appropriate level of abstraction to analyze and evaluate many important questions about software systems. Thereby, architecture offers the possibility to focus on the most relevant facts for a certain question and to abstract from less relevant facts. Additionally, architecture offers the advantage that architectural ideas can be made available quite early in the development lifecycle, which allows the analysis of crucial properties of the resulting systems, without the need to build them first.

Different types of methods

Different types of architecture evaluation methods exist which mainly differ with respect to the different prerequisites they require (point in time in system lifecycle, availability of artifacts and documentation, availability of resources and time for evaluation) and to the type of evaluation result (questions to be answered, required level of confidence) they can produce.

Scenario-based

A very widespread type of architecture evaluation is scenario-based evaluation. Therefore, important requirements are elicited from stakeholders and precisely expressed with so-called architecture scenarios [BCK03]. Then, evaluators and architects discuss together how the architecture at hand fulfills the scenarios and where there are gaps or risks in fulfillment. Well-known approaches of this type are SAAM (Software Architecture Analysis Method) [KAB96] and ATAM (Architecture Tradeoff Analysis Method) [CKK01] by the SEI (Software Engineering Institute). These methods can be applied at any level of detail of architecture specification. Of course, a low level of precision in input artifacts also results in lower confidence in the evaluation results. Scenario-based architecture evaluation methods like SAAM and ATAM are typically targeting at quality attributes in general, but do not provide specific support for quality attributes like flexibility.

Quality attribute-specific and more formal methods

For more confidence in the evaluation results, more detailed evaluation inputs and more focused evaluation methods are necessary. That means that typically formal architecture models have to be provided which cover specific information for the analysis at hand. For example that would mean that for detailed performance analyses architecture models must cover the necessary timing information. Many of the architecture evaluation methods for maintainability and flexibility as described in the following sections fall into this category. However, there is still a broad range in the degree of formalization. With a high degree of formalization, also automatic calculations and simulations [Bos00] become possible, as realized in this thesis.

A survey of architecture evaluation methods is published in [RG08].

3.3.2 Evolution Complexity

Eden and Mens introduced the term *Evolution Complexity* [EM06] following the idea of *Computational Complexity*. Thus, they introduce similar to the Big Oh notation a notation which expresses the complexity of changes in classes of growth; that means a change is independent of the system size ($O(1)$) or grows linearly with the system size ($O(n)$). They count the number of affected modules for a certain change scenario in a software system. However, there are basically only two complexity classes in evolution and thus these metrics are not accurate enough.

Consequently, in a second step they introduce more evolution metrics, for example depending on the number of lines of codes affected or on the cyclomatic complexity of the affected modules. In a number of case studies they apply their metrics on well-known Java design constructs or on architectural styles and calculate the metrics for assumed change scenarios.

Summarizing, they introduce interesting ideas for measuring flexibility but they do not provide a consistent idea of how to use these metrics and of how to embed them into engineering practices.

3.3.3 Analyzing Modifiability at Architecture Level

Two major (and intertwining) research streams in architecture-level modifiability analysis can be observed and are described in the following.

Lassing, van Vliet et al describe two larger case studies of evaluating flexibility of software architectures [LRV99a, LRV99c]. They base their analysis on scenario-based architecture evaluation methods [KAB96], but they do an explicit analysis of the quality attribute flexibility / modifiability and derived criteria. These criteria (Impact level in terms of components affected by change; Multiple owners; Arising conflicts from multiple ver-

sions of software) are used to evaluate how flexible a system is with respect to a particular scenario. For their case studies, they describe the architecture of the systems (system itself (micro architecture) and system in context (macro architecture)) and discuss for a set of elicited flexibility scenarios how well they are supported by the architecture.

Bengtsson and Bosch developed a method “Architecture Level Prediction of Software Maintenance” (ALPSM) [BB99], which aims at analyzing maintenance effort during architecture design. It is a scenario-based approach as well and introduces a weighting of change scenarios and an estimation of component sizes. In a step “scripting the scenarios”, the change impact of scenarios is analyzed and in a final calculation the average size (in LoC) of a change is derived, from which, with several assumptions, maintenance efforts can be derived.

In the following, Lassing, van Vliet, Bengtsson, and Bosch published together and called their method “Architecture-Level Modifiability Analysis” (ALMA) [BLB+00]. The core of the method is scenario-based, aiming at the analysis of change impact of anticipated scenarios. In [LRV01], architectural viewpoints that provide information for modifiability analysis are introduced (context, technical infrastructure, conceptual, development viewpoints) which represent a meta-model of architecture.

Refinements, more case studies, and experiences with ALMA have been published in [BB01, Ben02, LBV+02, BLB+04]. This thesis bases on several ideas introduced in ALMA, e.g. the weighting of scenarios and the calculation of impact sizes. While ALMA puts more focus on the calculation of maintenance efforts and the process of scenario elicitation, we put more focus on the integration of flexibility evaluation in the architecture construction process and in particular on more explicit separation of business logic and infrastructure. Additionally, we provide a notation for modeling change impacts as part of the architecture model. This allows automatic calculation of flexibility metric values in architecture modeling tools.

3.3.4 Modifiability and Real Options Theory

Bahsoon and Emmerich developed an approach to calculate the value of investing into architectural flexibility, which is called “ArchOptions” [BE03, BE04, Bah05, BE06]. They put architectural stability [Jaz02] as a major goal as it leads to moderate cost for occurring changes. In order to achieve architectural stability, the architecture has to be flexible enough to absorb the changes. They developed a model based on real options theory: Put simply, they see investing into flexibility similar to buying real options which allow conducting a certain change at a certain later point in time at a certain price. They found analogies for typical parameters in real options theory and thus can use the calculation models provided by the model they used. Consequently, they provide interpretation guide-

lines which allow a judgment on whether the investment into flexibility for certain changes is worthwhile or not.

There is little overlap of this work on real options theory and the work of our thesis. We do not emphasize the ratio of investment into flexibility and the payoff in detail; thus this work complements our approach well.

3.3.5 Palladio and Maintainability Prediction

At Karlsruhe Institute of Technology (KIT), the *Palladio Component Model* (PCM) [RBB+11] was developed. It is an architectural framework providing methods for architecture definition and analysis. The basis of the framework is an architecture meta-model and a distinction of roles involved in development and their relationship to the models. Architecture modeling and architecture analysis based on PCM are supported with Palladio Bench, an Eclipse-based tool-suite [Palladio].

Palladio supports architecture analysis for several quality attributes like performance, reliability, or maintainability. In [BKR07, BKR09], they describe how PCM is utilized to represent the relevant information for performance prediction and how the model can be analyzed. For maintainability, the *Karlsruhe Maintainability Prediction* (KAMP) [SR09] approach is defined. It is also based on PCM and supports the calculation of change effort for certain anticipated change scenarios. A detailed change impact analysis for a scenario is conducted which can be calibrated with bottom-up effort estimations for conducting the changes.

KAMP bases on some ideas about the analysis of maintainability that are similar to the ones of our thesis: It works in a scenario-based way and calculates maintainability based on impacted architectural elements. While KAMP prescribes in more detail how to take architectural elements like components and interfaces into account, our approach stays rather general and allows including any architectural element as needed by the architect. KAMP supports the automatic derivation of change impact by model comparison whereas our approach targets at a light-weight and more abstract modeling of change impacts by the architect during architecture design. KAMP rather supports the execution of changes at a certain point in time by estimating the change effort and deriving work plans for the change whereas our approach targets at the construction time of the architecture when the flexibility needed is built in. Furthermore, KAMP takes activities like deployment into account for the effort estimation.

3.3.6 Enterprise Systems Modifiability Analysis

Lagerström, Johnson et al developed an approach for analyzing the modifiability of enterprise systems; they target at application landscapes

with an enterprise-wide focus. First, they developed so-called *Extended Influence Diagrams* [PLN+07] as a basis for the expression of their meta-models. In [Lag07], the first meta-model for maintainability is developed. Thereby, maintainability is seen in our broadest sense, comparable to what we called “Agility” in Section 3.1.1 (covering aspects of Personnel, Process, Documentation, Architecture Quality, Platform Quality, Source Code Quality). As a quantitative basis for the meta-model, Probabilistic Relational Models (PRM) are used. The analysis is done scenario-based and the evaluation results are probabilistic values, too.

In [LFJ+09], a method for creating enterprise architecture meta-models is introduced, which is then applied for modifiability (renamed from maintainability). The resulting models have a qualitative part with elements, attributes, and causal relationships (see Figure 23) and a quantitative part which contains probabilistic calculations for the derivation of the overall modifiability or cost values. A more detailed meta-model is presented in [LJE10], which is dedicated to software change cost estimation. It describes different views of the meta-model (organizational, project, documentational, system) and hierarchical views for characterizing system parts in order to control the model complexity. In [LJH10], the authors summarize the evaluation models and methods and illustrate their usage with case studies.

The approach of Lagerström and Johnson differs from our approach mainly in the coverage of aspects concerning changes. Whereas they try to cover all relevant aspects influencing modifiability in the broadest sense, we aim at a much smaller scope, namely at the impact of architectural decisions on flexibility, and thus achieve a higher accuracy. Further, we also support constructive aspects. An approach like the one dis-

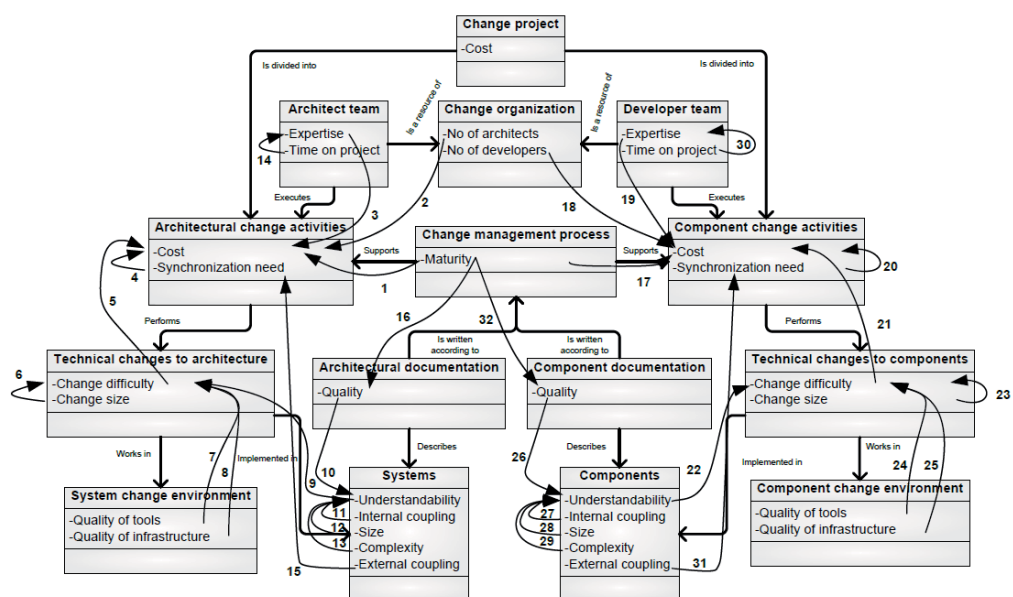


Figure 23: Modifiability meta-model [LFJ+09]

cussed is helpful in the sense that it covers the relationships among different types of influence factors and allows identifying where investment into improvements is promising. It could be combined with our approach in the sense that our approach delivers more accurate evaluation results for a small excerpt of the overall evaluation.

3.3.7 Further Related Research

In this section, we present some more related individual research results.

In [ZYX+02] an approach for change impact analysis at the architectural level is introduced in order to support evolution. This approach defines the notions of *slicing* and *chopping* at the architectural level. The main usage from a change impact analysis perspective is to identify in an architecture how changes propagate to other components in case a certain component has to be changed. Doing so does not take the relationship to change requirements into account but works at architectural elements only. Consequently, this approach is complementary to the approach introduced in this thesis: It could be used for auto-completion of change impact views after adding first impacted architectural elements.

In [AFL+05], an experiment on the exploitation of flexibility is described. They have several findings: developers not always make use of built-in flexibility; even sophisticated flexibility mechanisms might not be recognized and used; their conclusion is that besides a sound documentation also the intention behind flexibility mechanisms has to become clear. Further, deterioration of architecture has to be prevented to benefit from flexibility in the long run.

Sneed provides a method for estimating maintenance cost [Sne95]. For a specific change he also analyzes the size of the so-called *Impact Domain* and applies several adjustment factors like the complexity factor, the quality factor, and the project influence factor. Finally, the effort for the change is derived from a maintenance productivity table. The approach is more related to cost estimation work than to our approach; however, it uses similar mechanisms for the impact estimation and the variation with influence factors.

3.4 Flexibility in SOA-Based Information Systems

Flexibility is often promised and expected as one of the key benefits of SOA (see Chapter 1). However, there is not much research on this topic. One publication with a strong focus on business flexibility and some architectural aspects is [Spr05]. Mostly, when it comes to the idea of flexibility in SOA, runtime adaptation is in focus, as described in [Smi08].

SOA as an architectural style with some implications for flexibility is described by IBM in [Lub07].

A SOA research agenda has been published under the supervision of SEI and others [LSK07]. It identifies among many other research questions also those that are closely related to our thesis. In the *Engineering Domain, Architecture and Design* and *Maintenance and Reengineering* have been identified. With respect to Architecture and Design, we provide our architecture approach for flexibility as well as the SOA-specifics in Chapter 6. With respect to Maintenance and Reengineering we provide proactive tool-support for maintenance activities and impact analysis, mainly by the change impact views and our tool support.

As architecture mechanisms for flexibility in SOA, ESB (see Section 2.2) [Cha04], BPM [Wes07], or BRM [BS09, End04] are often found. However, these descriptions typically describe only the architectural mechanisms and not how to use them in terms of business logic mapping for achieving *True Flexibility*, as aimed at in our thesis.

Another branch of research that is related to BPM deals with process flexibility [RSS06, SMR+08, Kan09, Kan10]. This typically means that processes which are supported by workflow languages and engines can be changed flexibly, even during the runtime of the system. This type of flexibility is out of scope of this thesis.

3.5 Summary and Conclusion

In the previous sections, we described related work to all our research directions as sketched in the introduction of this chapter and provided one-to-one comparisons. In this section, we summarize the analysis of related work and the gaps to be filled by this thesis.

It turned out that there is no single comparable approach, which is supposed to be improved in this thesis. Rather, for all the research directions, there are many related ideas and approaches which provide partial foundations for our thesis. This thesis contributes mainly in the area of design for flexibility, with support of analytical methods and refinements for SOA-based systems. Therefore, a consistent and crosscutting approach around the quality attribute flexibility is introduced.

The center of this thesis is the constructive support for flexibility. As described in Section 3.2, current architecture definition approaches in general and SOA design approaches in particular do not offer this support. We introduce a more detailed process and guidance on how to achieve flexibility in combining the application of adequate architecture mecha-

nisms (based on quality-driven design) and business logic mappings (based on functional decompositions).

A key idea of our approach is to support an architect with immediate feedback on the flexibility of the architecture under construction. Most of the related work we found is on evaluation of flexibility or similar quality attributes at architecture level (see Section 3.3). Consequently, there is a strong foundation for our analytical support. However, the approaches found still require manual analysis and interpretation for flexibility evaluation, which is not desired in our approach. Therefore, we need an idea of how to achieve an automated analysis of flexibility.

As a basis for our methodical contributions, we need a consistent foundation in terms of defining flexibility and the relationship to architecture. Thus, we reviewed related work on flexibility and related quality attributes from different domains (see Section 3.1). This related work gives a solid foundation to synthesize a consistent meta-model of flexibility at architecture level, similar to what others did for different scopes.

Finally, we explored how flexibility is addressed in research for SOA-based systems (see Section 3.4). There we found that although many architecture mechanisms with inherent flexibility potential exist, it is not clear how they are to be selected and utilized in architecture definition. This is the gap we fill there.

From our research directions (R.D1 – R.D4) and the related work we derive the following research challenges (see Section 1.4):

- **R.C1:** How can the relationship between flexibility and architecture be precisely characterized and how can this be used for 1) better elicitation of flexibility requirements, 2) more guidance for architecture design, 3) measurement of flexibility?
- **R.C2:** How can an architecture construction process support architecture design for flexibility with appropriate architectural mechanisms and business logic mappings?
- **R.C3:** How can the flexibility of an architecture under design be automatically predicted for near-time feedback on flexibility to an architect?
- **R.C4:** How should architectural information about paradigms / technologies like SOA be described and used in architecture construction in order to exploit their flexibility potential?

As described in Section 1.4, the research ideas **R.I1 – R.I4** are followed (see Figure 24) in order to address the research challenges.

In summary, we provide an approach for guiding architecture definition towards flexibility. One aspect is direct feedback about the achieved level of flexibility to the architect, which requires extending existing flexibility

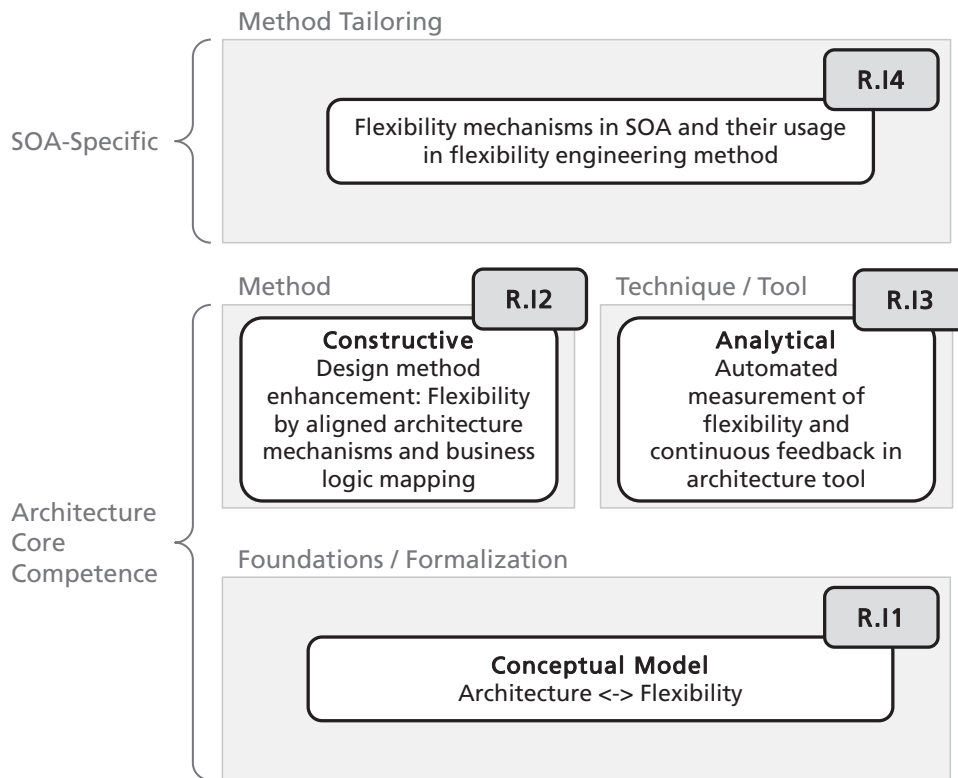


Figure 24: Overview on research ideas of the thesis

evaluation methods towards automated evaluation. As a basis we need a consistent meta-model of flexibility at architecture level. Finally, we contribute support for making best use of typical SOA architecture mechanisms in achieving flexibility.

4 Flexibility as a Quality Attribute of Software

"Flexibility is the root of all evil."
Scott Jenson

Scott Jenson writes in "The Simplicity Shift" [Jen02]: "Flexibility is the root of all evil". Although not written for software systems and their internal organization but rather for industrial product design and the related user interfaces, this citation and its explanation hold also for flexibility of software: Flexibility per se is not evil, but trying to make everything flexible leads to poor design. On the other hand, flexibility can be very valuable when fitting the needs. We will describe these ideas more formally and grounded for the world of software in this chapter.

In this chapter, we provide an elaborate definition and discussion of characteristics and aspects around flexibility. Our definitions are based on existing definitions and concepts but aim at more depth in clarifying concepts and their relationships. We introduce or reference definitions of the key terms in this chapter. Some of the definitions have already been introduced in Chapter 1 for precisely describing the problem addressed in this thesis.

In [SHN01] (see Section 3.1.3), flexibility research in the area of space systems is introduced. Questions are posed which also guide our research well. These questions will be addressed in this and the next chapter.

- "What is flexibility? How does a formal definition look like?"
- "Why or when is flexibility needed in system design?"
- "How can one design for flexibility? What are the design principles?"
- "What are tradeoffs associated with flexibility?"

First, we start with a characterization of flexibility in Section 4.1. Then, we elaborate on flexibility requirements in Section 4.2. For the achievement of flexibility, the role of architecture is discussed in Section 4.3. Finally, all concepts are summarized and formalized in a conceptual model for flexibility in Section 4.4.

4.1 Characterization of Flexibility

In this section, we characterize flexibility. First, principle characteristics are sketched (4.1.1). Second, flexibility is viewed in the perspective of the software product lifecycle (4.1.2). Then, the relationship of flexibility to software engineering artifacts is outlined (4.1.3). Finally, flexibility is described in a spectrum of uncertainty (4.1.3).

4.1.1 Principle Characteristics

Flexibility is always about **changing software in the future**. In practice, software is often expected to support any type of change, as it is inherently “soft”. Theoretically, most changes can be conducted but the effort for conducting a change can be enormous. Thus, flexibility always means to support changes with little effort and acceptable cost.

Definition 9 *Software Change (or simply “Change”)*

A change of a software system is a change of implementation artifacts of the system (and consequently the executable system) in order to fulfill changes in the system’s set of requirements.

Flexibility is always about **future, anticipated requirements** to a system which have not been realized yet and can only be realized via changes to the system at **development time**. **Keeping effort for changes little** requires that only local changes to a few implementation artifacts are necessary. The realization of a software system always leads to the situation that certain requirements have encapsulated realizations while other requirements have crosscutting realizations (determined by the architecture and the nature of requirements). Consequently, there are always potential changes to requirements which cannot be conducted with little effort, which means that a **system cannot be flexible with respect to all changes** of requirements. This is depicted in Figure 25, where all potential changes to requirements are sketched with an assumed effort for conducting the changes. Thus, most approaches towards flexibility (see Chapter 3) state that flexibility of a system **can only be judged with respect to certain anticipated changes**. Such anticipated changes are expressed as **flexibility requirements** (see Definition 3). On the other hand, for the same reasons there are changes in each system that can be easily conducted. For these, the term **flexibility potential** (see Definition 5) is introduced. Only if the changes expressed in flexibility requirements are covered by the changes supported by the flexibility potential, the system has valuable flexibility (= **true flexibility**, see Definition 7).

Besides the effort for **changing implementation artifacts** (as mentioned above), changing a software system leads to effort caused by

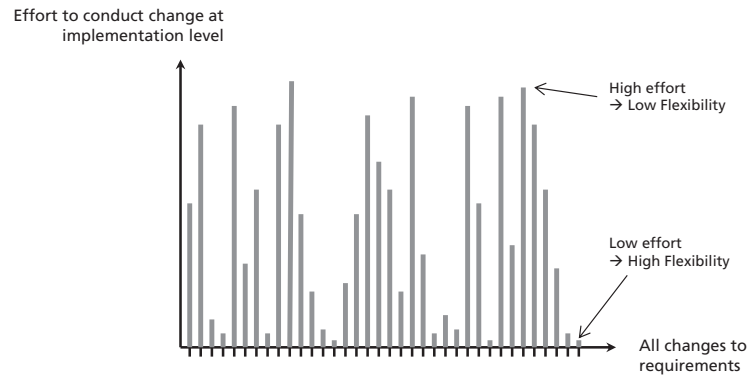


Figure 25: Distribution of change effort to change requirements

other activities like first analyzing and understanding the system, testing the system after change, putting the system into production after change, etc. However, these efforts are related to the quality flexibility as defined in this thesis, but they are expressed in other quality attributes like understandability and maintainability, as delineated from flexibility in Section 3.1.1. Please note that this is the definition for this thesis and it is not commonly agreed on as described in Section 3.1.

As flexibility requirements are **requirements about changes of requirements** in the future, there is an inherent indirection, which makes flexibility in practice often harder to understand, to construct for, and to evaluate than other quality attributes. An additional complexity is introduced as flexibility requirements are often not sharp and clearly defined but rather fuzzy (see Section 4.2). Definition 10 gives a summarizing definition of flexibility covering the aspects discussed. A more formal and measurable definition is described in Section 4.3.3.2.

Definition 10 Flexibility

Flexibility is the property of a software system to allow conducting certain anticipated changes to the system (expressed in flexibility requirements) with acceptable effort for modifying the system's implementation artifacts. This means that the flexibility requirements are covered by the flexibility potential of the system.

Consequently, when we use the term flexibility, true flexibility is meant. True flexibility is thus an additional term that was introduced to make the distinction between flexibility and flexibility potential clear. Actively designing for flexibility means to come to a situation where only little effort is needed for important changes and high effort is needed for changes never conducted, rather than having a random relationship.

An architect aiming at flexibility has to accept limitations with respect to flexibility requirements that can be covered. This has several reasons for this:

- A system cannot be flexible with respect to all flexibility requirements at the same time (see arguments above)
- Introduction of flexibility often comes with additional cost (see Section 4.3.4)
- Flexibility typically requires tradeoffs with other quality attributes (see Section 4.2.3)

4.1.2 Flexibility in the Software System Lifecycle

Flexibility affects the **whole life cycle** of a software system from initial development to retirement. As described, flexibility is the property of a system to support changes to a software system at acceptable cost. Changes to a software system typically occur during maintenance activities or also in later phases of initial system development. In order to benefit from flexibility later on, it has to be designed and built into a system earlier. Designing and building in flexibility is very typical during initial system development, but it can be also done in larger maintenance projects when the need for flexibility is recognized. We distinguish mainly two high-level activities around flexibility (see Figure 26):

1. **Designing and building in** flexibility
2. **Exploiting** flexibility

Both activities can basically happen at all times in the system lifecycle. Of course flexibility has to be first built in with respect to a particular change, otherwise exploitation is not possible. This thesis focuses on designing and building in flexibility (see Chapter 1), whereas the exploitation of flexibility is not covered.

Looking at **flexibility requirements** is possible from two points in time in the system life-cycle. From the **a-priori** point, flexibility requirements have been anticipated and weighted during system design and building. From the **a-posteriori** point, which means looking back at the changes conducted in a system, all actual change requirements can be listed and

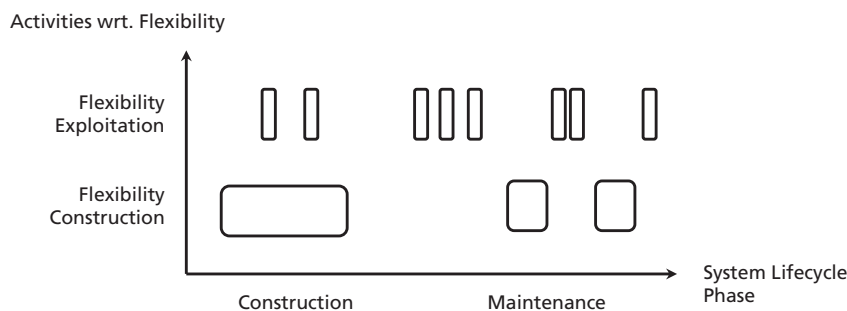


Figure 26: System lifecycle phases and activities related to flexibility

also the effort to realize them is known. For our method and thesis, mainly the a-priori perspective is relevant.

4.1.3 Flexibility and Software Engineering Artifacts

Flexibility has a relationship to artifacts along the software engineering process. Thus, we sketch the role of requirements, architecture, and implementation for flexibility.

Require- ments	Flexibility requirements are non-functional requirements as many other requirements. A specific aspect is that they express some indirection: They are requirements about changing the set of requirements at a certain point in time in the future. Consequently, flexibility requirements are elicited and treated similar than other requirements. Guidelines for their elicitation and description are outlined in Section 4.2.
Implemen- tation	According to our definition of flexibility (see Definition 10), flexibility means that changes can be conducted with acceptable effort for modifying implementation artifacts. The key assumption behind this is that most of the effort for modifying system artifacts has to be spent on the implementation level. With implementation level we basically mean all artifacts that are developed to become part of the executable system, for example source code files in any programming language (e.g. Java, C, C#, Fortran files, etc.), scripts and descriptors as parts of technologies (like SQL queries, Java EE deployment descriptors, XML-based files for configuring Spring or Eclipse, etc.), or database definition schemas. In the context of model-driven development [Sel03], such implementation artifacts are often not directly created by programmers, but they are generated fully automatically from higher-level model artifacts. Interpreted in our understanding, the flexibility definition always refers to the last stage of implementation artifacts to which the modifications have to be done manually by programmers (because the update of the generated artifacts is very easy through automation). This can also be models. An example is the addition of a data field through all levels of software from database over the logic layer to the UI: If the system is described with a model that contains the data and can be updated and the respective code parts can be generated, there are a lot of changes to the code but only minimal effort for programmers, which results in high flexibility. Summarizing, we can say that flexibility is related to the artifacts that have to be changed by programmers or other engineers and cause the significant efforts.
Architecture	Referring to the definition of flexibility: Why is architecture important and why should we not care about the implementation level only? Architecture is the blueprint and manifestation of all design decisions which are realized at implementation level. Consequently, architecture is the abstraction used to design and measure flexibility without the need to consider every detail of the implementation level. In that sense, architec-

	Configurability	Flexibility	„Good Design“
<i>Level of uncertainty (change? probability?)</i>	Low (all changes anticipated)	Medium (anticipation of changes and probabilities)	High (no anticipation of changes)
<i>Principle how to design for change</i>	Build every anticipated configuration into the system	Build solutions that allow low effort for anticipated and likely changes	Use general design principles like „low coupling“, „high cohesion“
<i>Possibilities to measure adequate support for changes</i>	Check whether all anticipated configurations are built in	Check whether anticipated changes can be conducted with acceptable effort	Calculate metrics about design principles, like low coupling and high cohesion
<i>Principle how to realize a change</i>	Simply select the appropriate configuration	Development activities exploiting flexibility (low effort if change was anticipated)	Development activities, effort depends on how well change is supported by design

Figure 27: Flexibility in a spectrum of uncertainty

ture allows controlling the high complexity of today’s systems when talking about flexibility and it allows us to plan for flexibility at an early point in time in the development lifecycle. Another aspect of architecture in the context of changing a system is that the architectural model and documents might undergo changes, too. Our assumption is that changing architectural models and documents causes negligible effort compared to the effort for changing the implementation. One exception is in line with the discussion of model-driven development above: In case the architecture models can be directly used to generate the implementation artifacts, they are the artifacts that are changed by “programmers” and thus have to be considered for flexibility. Although the effort for changing a system is mainly caused at the implementation level, flexibility can be only designed for at architectural level, as flexibility requires the consideration of crosscutting aspects in software. Thus, we strongly focus on the role of architecture for flexibility (see Section 4.2.3) and on how to design flexible architectures (see Chapter 5), always maintaining the relation to the implementation level.

4.1.4 Flexibility in a Spectrum of Uncertainty

Flexibility always comes with uncertainty. Changes have to be anticipated to design for and it is not clear what exactly the changes will look like and whether they will really happen (probability). In this section, we look at situations where the uncertainty is not given and at situations where the uncertainty is very high. This spans a spectrum in which flexibility is located. The idea is to compare flexibility with these situations and learn about the differences. Figure 27 depicts all the aspects discussed.

Designing for change depending on uncertainty	<p>The situation with low uncertainty, in which changes can be anticipated rather easily, is typically addressed with <i>Configurability</i> [NM10], as often found in Product Line Engineering. That is, changes are well known and likely and can thus be built directly into the software system as alternative configurations. On the other end of the spectrum, uncertainty is high and no concrete changes can be anticipated. Of course, there is still the wish to be able to conduct changes at relatively low cost, but without having a clue of what they could be. Then, the only possibility for design is to follow best practices and design principles like “low coupling”, “high cohesion”, “encapsulation”, etc. (see also Section 4.3.1), which we call here “<i>Good Design</i>”. When designing for flexibility, an architect will probably apply the same or similar design principles, but target them explicitly at optimizing the anticipated flexibility requirements, whereas in the case of no anticipated changes the architect has to rely on his experience and to some extent luck. This can be discussed with Figure 25: In case of anticipated flexibility requirements, the architect can concentrate on those, neglecting the change effort for other changes. In case of high uncertainty, the architect has to balance and try to minimize effort at a more general level.</p>
Measuring support for changes	<p>Depending on the level of uncertainty, different types of measurements can be applied. In the case of configurability, it is quite easy to check whether all configurations are built in. For flexibility, the measurement is possible indirectly only, by checking whether the anticipated scenarios have limited impact. In the case of “<i>Good Design</i>”, it can only be checked whether the design principles are adhered to.</p>
Realizing changes	<p>When it comes to a change, in case of configurability the appropriate configuration can simply be chosen. In case of flexibility, the change is conducted with relatively low effort, provided it was an anticipated change. In the case of “<i>Good Design</i>”, it is to a greater or lesser extent by chance how much effort has to be spent on changes: It can be very low effort if the change fits the design decisions made, but it can also be high effort impacting large parts of the system.</p>
	<p>In [NM10], we discussed the differences between configurability and flexibility in more detail. The classification and distinction as introduced here is intended to understand typical situations and to know how to deal with them. Information systems in practice do not belong to just one of the three categories. Rather, they have aspects of configurability, flexibility, and “<i>good design</i>”, depending on the possibility to clarify uncertainties.</p>

4.2 Flexibility Requirements

The term *Flexibility Requirement* is defined in Definition 3 as “a requirement that expresses the potential need for changing the set of require-

ments of a software system in the future". A flexibility requirement is very similar to a *Change Requirement* (see also Definition 9). The key difference is that a flexibility requirement expresses only the potential need whereas a change requirement can be seen as an actual requirement that has to be realized. A term also often found is *Change Request*, which denotes a request to conduct a particular change requirement. In the perspective of the software system lifecycle (see Section 4.1.2), flexibility requirements occur in the activity of constructing for flexibility and change requirements occur in the activity of exploiting flexibility.

In the following sections, we describe how flexibility requirements can be captured with scenarios (4.2.1) and how scenarios can be characterized and classified (4.2.2). Finally, we give an overview on requirements typically competing with flexibility and requiring adequate tradeoffs (4.2.3).

4.2.1 Capturing Flexibility Requirements with Scenarios

For the evaluation and definition of software architectures, architecture scenarios [BCK03, CKK01, RW05] are a proven and established means to precisely express the architecturally-relevant requirements. Also for quality attributes related to change (maintainability, modifiability, etc.), architecture scenarios are strongly applied (see Chapter 3). Different names can be found in literature, e.g. *Change Scenario*, *Modifiability Scenario*, *Evolution Scenario*, which all have a similar meaning. In line with our terminology of flexibility, we use the term *Flexibility Scenario*.

Definition 11 *Flexibility Scenario*

A flexibility scenario is an architecture scenario expressing flexibility requirements from an architect's perspective.

By expressing standardized information like source, stimulus, environment, artifact, response, and response measure, architecture scenarios provide a frame for a detailed description of requirements. In particular the context information as the triggering stimulus or the environment in which the scenario takes place give an architect additional information to reason about. In that sense, flexibility scenarios mainly have the task to foster the precise and complete elicitation and description of flexibility requirements. Further, they fully integrate in the typical approaches of architecture definition and evaluation approaches.

Our definition says "... from an architect's perspective". This is an important aspect of flexibility scenarios, as they do not only focus on the problem space, but can incorporate knowledge about the solution space of a system can be incorporated as well (see Figure 28). Concretely, this means that in the formulation of flexibility scenarios intended or existing design decisions, components, or technologies can be mentioned to be as accurate as possible.

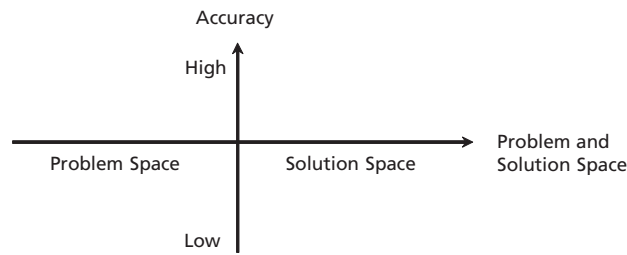


Figure 28: Characterization of architecture scenarios

Architecture scenarios in general are not expected to always express the same level of abstraction or accuracy (see Figure 28). Rather, they are used to express the information as it is necessary and possible in the concrete context. This particularly holds for flexibility scenarios. Inherently, there is a certain level of fuzziness for future changes, which results in the accuracy of the scenarios and the assumptions about probabilities of scenarios. Depending on the level of accuracy of flexibility scenarios, the construction and evaluation activities can be more or less focused.

In the following section, we describe how flexibility scenarios can be further characterized and classified in order to elicit and document the most relevant information for flexibility construction and evaluation.

4.2.2 Characterizing and Classifying Flexibility Scenarios

Characterizing a quality attribute in detail has the main intention to make previous experiences about challenges in designing software systems explicit (see also Section 2.1.3). For the quality attribute *Modifiability*, which, according to our classification, is very similar to flexibility (see Figure 22), a characterization is given in [BCK03]. There, modifiability is characterized along the standard scenario template (source, stimulus, artifact, environment, response, response measure). Further characterizations of scenarios can be found in [BLB+00, LRV99b], which mainly characterize along the requirements for change and the types of change impact from an architectural perspective.

We base our characterization on the standard scenario template [BCK03]. Our contribution extends the characterization of modifiability with typical questions and sub-characteristics to be asked to elicit the respective information (see Table 2), and typical values for the characteristics (see elaborations of the questions).

<i>Scenario Template</i> <i>[BCK03]</i>	<i>Questions</i>
Source	Who or which event triggers the change?
Stimulus	How likely is the change? How often does the change occur?
Artifact	What has to change? Which concrete business logic aspects are related to the change?
Environment	Who makes the change? When is the change made?
Response	What is impacted by the change from an architecture perspective?
Response Measure	How much effort is needed to change existing parts? How much effort is needed to conduct the change?

Table 2: Scenario characteristics and questions

As typically done for architecture scenarios, there is no need for a tabular representation. Rather a pure textual representation focusing on the key information parts is more appropriate for an architect’s work.

In the following, we provide for all questions typical values as experienced in projects or derived from literature [BCK03]. This overview of typical values is not intended to be complete but covers experiences that are useful when eliciting and documenting flexibility scenarios.

Who or which event triggers the change?

A change to a software system is typically triggered by a stakeholder of the system or initially by an event which makes stakeholders triggering the change.

Typical stakeholders: manager, customer, user, architect, developer

Typical events: merging of organizations’ IT, migration or exchange of IT systems, business process improvement activities, integration of systems with external systems, change of legal or other regulations, availability of new technologies

How likely is the change?

Flexibility requirements are defined as potential changes, thus there is always some uncertainty whether the change will be conducted in the future. As the probability of different potential changes can strongly

vary, it is an important information for an architect to know the probability of a flexibility scenario in order to balance architectural solutions.

How often does the change occur?

The expected frequency of changes is also an information that should be provided for an architect in order to allow adequate support for frequent changes.

What has to change?

The key aspect of a flexibility scenario is to describe which aspect of a software system has to be potentially changed. As there are different types of requirements, these types can also be affected by changes.

Business Logic or functionality: Changes of functions, data, processes, UI, etc.

Quality: New quality attributes, qualities in new context, change of quality level, new capacity levels, etc.

Technology: Integration or replacement of technologies, usage of new features of used technologies, etc.

External systems: Integration with new or other external systems, changes due to changes in external system, etc.

Which concrete business logic aspects are related to the change?

Changes are often described at a level relating to realization concepts but not related to concrete business logic (functionality). An example change would be to say: "The order of steps in a business process has to be changed". As analyzed in Section 1.3, this change could have very different consequences depending on the concrete steps impacted.

Thus, we make this distinction explicit in our model by introducing two levels to denote whether concrete business logic is referenced (*Business Logic Specific (BLS)*) or no concrete business logic is referenced (*Business Logic Agnostic (BLA)*). Typically, this distinction becomes meaningful when changes aim at quality aspects, technology aspects, or the integration with external systems. When changes aim at functionality, they are often BLS by nature.

It can be valid to specify flexibility scenarios at both, BLS or BLA, levels. However, when specifying at the BLA level, typically the accuracy of the change impact estimation is much lower and constructive activities might aim at further refining of the scenario.

Who makes the change?

Each change has to be conducted by a particular stakeholder to some implementation artifacts of a software system (according to our definition of flexibility).

Thus, the typical stakeholder to conduct a change is a software developer. Nevertheless, the characterization can be more detailed: Developers can be in different *organizations* (e.g. in the case of sub-contracting or partnering), or different *skills* might be needed to conduct a certain change. Furthermore, easy changes to implementation artifacts like descriptive files could also be done by administrators or even skilled users.

When is the change made?

The time a change is made and applied to a system is also called *Binding Time*. According to our characterization of flexibility, this is typically at development time. Depending on the types of implementation artifacts changed (e.g. descriptive configuration files), some changes can also be conducted at installation time, but typically not at runtime.

What is impacted by the change from an architecture perspective?

The impact of changes from an architecture perspective (see Section 4.3) are the most solution-oriented aspects in the description of flexibility scenarios. The scenario description can range from describing no impact at all up to describing exactly the architectural elements that might be impacted, depending on the intention of the architecture scenario.

Typical architectural elements potentially impacted: Modules (or components) [functionality, process, data, UI, business rules], connectors, infrastructure and technology elements

Further architectural aspects potentially impacted: Architecture decisions in general, integration with external systems, deployment

The change impact description is closely related to Section 4.2.3, which in depth describes the role of architecture for flexibility.

How much effort is needed to change existing parts?

A change to a system can mean either to change existing parts of the system or to create new ones. It might be acceptable to invest quite an amount of effort to create new parts, but it might be expected to have as little impact as possible on the existing parts. Thus, we distinguish two aspects of effort needed for a change: The effort spent to change existing parts (this question) and the effort spent to create new parts (the next question).

How much effort is needed to conduct the change?

The overall effort to conduct a change might be also relevant since time and budget under certain circumstances might be strongly limited. Thus, it might be necessary to invest more upfront to allow a cheaper conduction of change later, during exploitation of the flexibility.

According to the characterization of elicited scenarios in the outlined categories, flexibility scenarios can be classified to give an architect better overview and easier access to the scenarios.

Section 5.2 briefly describes how to elicit flexibility scenarios and in particular how to make use of the characterization given here.

4.2.3 Flexibility and Competing Requirements

Flexibility as a quality attribute has to be balanced with competing requirements by finding adequate tradeoffs [BCK03, CKK01]. In this section, we sketch typical requirements competing with flexibility.

First, flexibility requirements are in competition with **other flexibility requirements**. Achieving flexibility always follows the strategy to localize change impacts: this results in certain architecture decisions about the modularization of the system. Different flexibility requirements might need different modularizations which cannot be realized at the same time. Second, there are other **development time quality attributes** like analyzability or testability which might be adversely impacted by flexibility. As flexibility often introduces architecture mechanisms with indirections, the system becomes harder to understand and to test. Third, flexibility can be competing with **runtime quality attributes** like performance or security. Depending on the architecture mechanisms introduced, additional indirections can lead to adversely impacted timing behavior or they might introduce security risks caused by additional technologies. This list is not intended to be complete, it rather sketches the different areas of potentially competing requirements. A well-founded analysis of tradeoffs can only be made at the level of concrete scenarios, describing flexibility and other quality attributes.

Although this thesis strongly focuses on flexibility from an architectural and methodical perspective, this does not judge the relative importance of flexibility compared to other requirements. This has to be balanced in the context of a concrete system (landscape) and thus architecture approaches have to provide similar support for all quality attributes.

The following section looks at flexibility from the architecture perspective and elaborates how an appropriate architecture makes a system flexible.

4.3 The Role of Architecture for Flexibility

In Section 4.1, we contrasted architecture and implementation as software engineering artifacts with respect to their role for flexibility. In this section, we further elaborate the analysis of the role of architecture for flexibility. This also covers the clear distinction of three different facets of architecture.

Architecture as design decisions and abstraction

We defined the flexibility of a system by the effort needed to change the implementation of the system to accompany the respective change (see Definition 10). How much effort is needed for the change mainly depends on how locally the changes can be conducted. This locality of changes is mainly determined by the design decisions made about the system, implicitly or explicitly. The entirety of principal design decisions exactly makes up the architecture of a system [TMD09]. In that sense, the architecture of a real system, comprising all principal design decisions, strongly determines the flexibility of the system, although the main effort for changes is at the implementation level (see Section 4.1). In Section 4.3.1, we present a closer look at different types of design decisions and at how they contribute to making a system flexible with respect to particular flexibility requirements.

Architecture as engineering artifacts

The entirety of principal design decisions about a system is an abstraction that can be documented as architecture models (e.g. represented as architectural views) and architecture documents for a software system. Then, architecture becomes a set of software engineering artifacts which are necessary to use the architecture for reasoning, analysis, or communication. Architecture models and documents are a means to achieve flexibility, as they allow architects to make explicit decisions for flexibility and to analyze these decisions early in the development lifecycle before the system has to be implemented. As described in Section 4.1.3, the effort for changing architectural models compared to the effort for changing the implementation is assumed negligible.

Architecture as engineering activities

A third facet of architecture is the one as an engineering activity, also called *Architecting*. In that sense, architecture comprises all activities that deal with definition, evaluation, or communication of architectural artifacts. A key task of architecting is architecture decision making; in the context of flexibility that means to make decisions that make the system flexible with respect to the flexibility requirements. In Chapter 5, we describe our architecting approach for flexibility. Systematic construction of flexibility is only possible at architectural level because global design decisions might be necessary to localize changes, which is not possible after distributing development to different development teams. Thus, this thesis focuses on architecture to constructively achieve flexibility. Section 4.3.2 explicitly describes how architects work to achieve flexibility, Section 5.4 introduces our analysis approach of flexibility to guide architects with short evaluation and feedback cycles towards a flexible architecture.

In this section, we first discuss which architecture decisions make a system flexible (4.3.1) and how an architect has to act to make a system flexible (4.3.2). Then, we describe an architecture meta-model covering the appropriate information for constructing and analyzing flexibility at architecture level (4.3.3), together with metrics for flexibility. Finally, we sketch cost considerations about flexibility (4.3.4).

4.3.1 Which Architecture Makes a System Flexible?

An architecture makes a system flexible when it prescribes design decisions that allow the flexibility requirements to be conducted with acceptable effort. Independent from a concrete flexibility requirement, similar strategies and architecture principles are available for realizing the needed flexibility potential. According to our definition of flexibility (see Definition 10), the effort to conduct the change described by a flexibility requirement has to be acceptable. How much effort is acceptable might vary considerably in the concrete context and cannot be defined in general. As the principles to achieve flexibility remain the same, we make the following simplification: We assume ideal flexibility to be given if a change proposed by a flexibility requirement can be realized with nearly no effort, meaning to change only a few lines of code.

The key strategy to achieve flexibility is to minimize and localize the change impact of anticipated flexibility requirements [BCK03]. Localization of change impact is achieved with architectural decisions that organize the overall implementation in a way that exactly the anticipated changes have only local impact. In particular, that means that a change does not require to revise key decisions, as they typically manifest over a larger extent of the system. Design decisions for minimizing change impact do not require to be invented from scratch for each new system. Rather, there are many architecture principles and mechanisms that give high-level guidance for making architecture decisions.

In this section, we start with an overview of general architecture principles and their support of flexibility (4.3.1.1). Then, we outline concrete architecture mechanisms supporting flexibility (4.3.1.2). Finally, we elaborate the interplay of architecture mechanisms and business logic mapping for flexibility (4.3.1.3) which we identified as a critical factor for flexibility in Section 1.3.

4.3.1.1 Architecture Principles Supporting Flexibility

Architecture principles are very fundamental ideas that are applied in the design of software systems. The term architecture principle is not commonly agreed on, but the principles we list are widely-known and mentioned in literature (see also Section 3.2). The following list of architectural principles is not necessarily complete and the principles are not al-

ways orthogonal or disjoint. Rather we list these principles to clarify the relationship of widely known principles and flexibility.

<i>Architecture Principle</i>	<i>Relationship to Flexibility</i>
Abstraction / Generalization	Abstraction allows handling common aspects with localized solutions
Indirection	Indirections allow to concentrate the impact of changes to a dedicated software module, which is addressed and included via the indirection
Information Hiding	Information hiding allows to encapsulate a certain aspect or internals of a solution in a software module, which allows localized changes of internal aspects
Loose Coupling	Loose coupling allows to reduce assumptions about other system parts and thus localizes change impacts
Low Coupling & High Cohesion	Low coupling and high cohesion of software modules allows localization of changes similar to loose coupling and information hiding
Modularization	Modularization allows separating different functionalities and system aspects into different modules so that module-internal changes can be handled locally
Separation of Concerns	Separation of concerns allows in the broadest sense the separation of different aspects of software in different realization units. By appropriate separation criteria, changes can be localized.

Table 3: Architecture principles supporting flexibility

The following architecture mechanisms supporting flexibility build on these architecture principles, too.

4.3.1.2 Architecture Mechanisms Supporting Flexibility

Architecture mechanisms (see Definition 4) are introduced into architectural designs in order to address certain requirements. For flexibility, there are many supporting architecture mechanisms available. Often, architecture mechanisms come in form of architecture styles, tactics, or patterns, which are proven best practice solutions for recurring design challenges. Whereas we introduced architecture mechanisms at a very high level of abstraction in Section 3.2.3, we give now classes and examples of architecture mechanisms and explanations. As for the architecture principles, this is not intended to be a complete list, but rather to give an introduction into flexibility mechanisms.

<i>Architecture Mechanisms</i>	<i>Example and Explanation</i>
Virtual Machines	Virtual machines are a very powerful and diverse mechanism to achieve flexibility. The basic idea is that an infrastructure component interprets some content in order to realize a behavior. Well-known examples are business process engines or business rule engines. Both types realize an externalized description of business process or business rules, which aims at easy and localized changes. The executing infrastructure components and the using components can remain unchanged.
Programming Language Mechanisms	Programming languages offer mechanisms like polymorphism or generics which can be used to build abstractions from certain system aspects and define a common behavior. This allows on the one hand easily changing the common behavior and on the other hand easily adding new classes for which to change the behavior.
Generation Approaches	With generation approaches, a meta-level of implementation is introduced, for example in model-driven development. With such approaches it becomes possible to localize change aspects that are not local according to the chosen decomposition of a system. For example, a data definition language can be introduced, from which data access components, all transport objects as well as the UI fields are generated. Then, it is very easy to introduce new data fields, although it potentially impacts all layers of a system.
Layering	The layered architecture style explicitly focuses on the separation of certain system aspects and on the limitation of relationships among the resulting layers. Depending on the flexibility requirements, changes can be localized to single layers.
PlugIns	PlugIns are a concept to achieve extensions of a system in an expected manner with nearly no change effort to the system (e.g. realized in Eclipse or Firefox). Thus, PlugIns are an example that brings very high flexibility and on the other hand require clear specification in terms of the potential changes being applied.
Service-Orientation	Service-orientation comes with a couple of mechanisms which are covered in detail in Section 6.

Table 4: Architecture mechanisms supporting flexibility

The architecture mechanisms as described can cover flexibility requirements concerning different system aspects like functionality, data, processes, or UI. Thus, the architecture mechanisms are typically specialized.

Applying the described architecture mechanisms often comes with the need for additional infrastructure components like business process engines or plugin frameworks. Depending on the needed degree of specialization, these infrastructure components can be individually developed or reused from available technologies. For example, many open-source or commercial business process engines are available that can be used in the development of a system for improving the flexibility.

4.3.1.3 Architecture Mechanisms and Business Logic

Flexibility cannot be achieved by only selecting appropriate architecture mechanisms supporting flexibility. As analyzed and described in Section 1.3, there is another type of architectural decisions which are crucial for flexibility: The decisions about the decomposition of business logic (or functionality) and the mapping to architectural elements and architectural mechanisms are decisive, too (see Figure 4). The actual flexibility potential (see Definition 5) of a system is always determined by the decisions made about architectural mechanisms and business logic mapping (see Definition 6). Consequently, the goal of designing for flexibility is to find appropriate combinations of flexibility mechanisms and business logic mappings to address the concrete flexibility requirements and to achieve what we call *True Flexibility*.

Business Logic is a broad term in this context. It subsumes the aspects Functions, Processes, Data, and UI of a software system. In particular, it aims at the concrete functional requirements of a system and the decisions how to decompose and realize these functional requirements.

Both, in the context of flexibility requirements and in the context of architectural solutions, descriptions with and without concrete business logic are possible and can be found in practice. Thus, we introduce a new terminology to distinguish whether flexibility is addressed at a level of detail only covering architecture mechanisms (Business-Logic-Agnostic, see Definition 12) or covering business logic mapping (Business-Logic-Specific, see Definition 13). An example of a BLA flexibility scenario is: "Change the order of process steps in a business process." An example of a BLS flexibility scenario is: "Change the order of Seating and Baggage in the CheckIn process." (see Section 1.3)

Definition 12 Business-Logic-Agnostic (BLA)

Flexibility scenarios as well as architectural solutions are called business-logic-agnostic if no description of concrete business logic is used to describe the flexibility scenario or the architectural solution.

Definition 13 Business-Logic-Specific (BLS)

Flexibility scenarios as well as architectural solutions are called business-logic-specific if concrete descriptions of business logic are included to

make the flexibility scenario or the architectural solution more precise. The amount and level of detail of business logic description may vary.

Further, we introduce a terminology to denote elements that are used at BLA or BLS level: Infrastructure Element (see Definition 14), Template Element (see Definition 15), and Business Element (see Definition 16). This distinction is aligned with other approaches like [Sie04], which also distinguish between business and infrastructure elements. The main extension we make is that we add a Template element, which is utilized during the design process as a placeholder for business elements and which explicitly allows making the distinction between BLA and BLS in architectural representations.

Definition 14 Infrastructure Element

An Infrastructure Element is an element which is introduced in the architecture of a software system in order to realize requirements that are typically non-functional.

Definition 15 Template Element

A Template Element is an element which serves as a placeholder for a business element during the development process, either in the requirements or in the solution. It represents typical properties of business elements but abstracts from the concrete business logic.

Definition 16 Business Element

A Business Element is an element which represents a business logic aspect of a software system, either in the requirements or in the solution.

An example of an infrastructure element is a business process engine, an example of a template element is a general service, and an example of a business element is a concrete service for seating in airline CheckIn.

Figure 29 summarizes the relationships of different element types and BLA, BLS, and BLM. The more business logic is described in flexibility

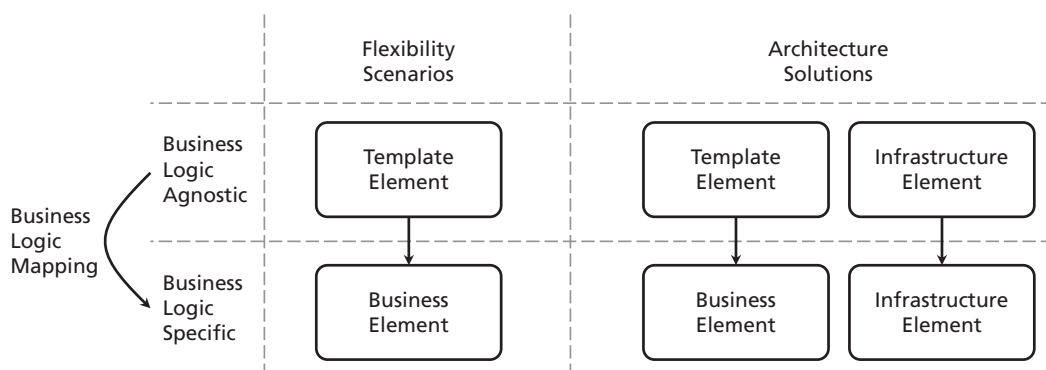


Figure 29: Distinguishing the levels business-logic-agnostic and business-logic specific

scenarios and architecture solutions the higher the accuracy and confidence of architecture-level evaluations and predictions of flexibility can be. That is, specifying flexibility scenarios and architectural solutions comes on the one hand with higher investments for creating the details, on the other hand it might return more accurate results. However, often it is not possible at an early point in time of system development to specify all business logic details for flexibility scenarios.

4.3.2 How Does an Architect Make a System Flexible?

An architect makes a system flexible by making appropriate architecture decisions which are sketched in the previous section. However, besides these decisions there are also the engineering activities and the artifacts produced and used by the activities. Architecting has different facets: Making decisions, modeling and documenting decisions, analyzing decisions for appropriateness. These facets can also be supported for flexibility with more detailed guidance. Chapter 5 describes our methodical contribution to engineering flexible systems.

A key benefit of making architectural decisions for flexibility is that the decisions are made at an early point in time in the sequence of development activities. At architectural level, decisions can be comparably easily evaluated and revised if they are not appropriate. By working at an abstract level, a lot of later rework effort can be saved. Thus, one of the key purposes (AEP, see Section 2.1.3) why to invest into architecture work is to predict the resulting flexibility properties of a system and avoid expensive rework. Another key purpose is to prescribe consistent and adequate decisions supporting flexibility, as potentially many developers have to derive an implementation fulfilling the flexibility requirements. In order to allow predictions of flexibility and consistent realizations, the architecture has to be made explicit in architectural models containing the adequate information to analyze flexibility and communicate the solutions. Hence, we describe an architecture meta-model covering this flexibility-relevant information in the next section.

4.3.3 Architecture Meta-Model and Metrics for Flexibility

After describing which design decisions are necessary to achieve flexibility and how to make these decisions, we describe in this section how architecture has to be modeled and represented in order to serve as a useful artifact in the process of defining a flexible architecture. Thus, we describe which architectural views capture the flexibility-relevant information (4.3.3.1). Then, we describe which metrics we introduce to analyze flexibility at the architecture level (4.3.3.2). Finally, we introduce a new architectural view, the change impact view (4.3.3.3)

4.3.3.1 Flexibility-Relevant Architectural Views

A system is flexible with respect to a certain change if the change can be conducted with only minimal impact on the implementation (see Definition 10). Thus, the focus of relevant architectural views is on the ones representing development time artifacts and properties of a system. In ACES-ADF, the *Development Time* dimension summarizes these relevant architectural views. However, our approach is not limited to a particular architecture meta-model; rather it is universal and can be used with nearly every architecture meta-model.

Modules	The key architectural element type with respect to flexibility is <i>Module</i> . A module is an abstraction of any development time or implementation artifacts. In literature, there is no common understanding of the term <i>Module</i> . Our definition is aligned with [CBB10]. Figure 30 sketches a meta-model with the key ideas about modules. A module can be a hierarchical <i>grouping</i> of other modules. Modules can have <i>uses</i> -relationships to other modules or to interfaces realized by modules. The uses-relationship is aligned with the definition in [Par79, CBB10].
Types of modules	Modules as described can represent and encapsulate different aspects of a software system, either in rather separated or in mixed form. The main aspects represented by modules in software are <i>functions</i> , <i>data</i> , <i>user interface</i> , <i>processes</i> , or <i>infrastructure</i> .
Realization of modules	Modules are architecture-level abstractions of implementation-level artifacts (see Section 4.1.3): Such artifacts can be diverse, depending on the decisions for implementation technologies. Examples of implementation-level artifacts are source code files in any programming language (e.g. Java, C, C#, Fortran files, etc.), scripts and descriptors as parts of technologies (like SQL queries, Java EE deployment descriptors, XML-based files for configuring Spring or Eclipse, etc.), or database definition schemas. Additionally, in the context of model-driven development, models used to automatically generate other implementation-level artifacts are considered implementation-level artifacts, too (see Section 4.1.3).
Role of technologies	Our meta-model does not assume any particular realization technology. Rather, implementation-level artifacts in any technology can be repre-

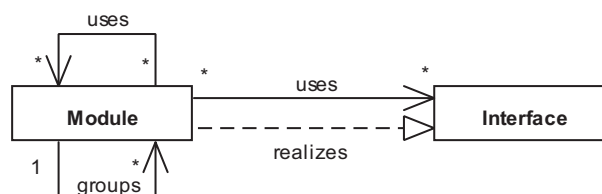


Figure 30: Architecture meta-model for modules

sented as modules. For convenience of architects and easier communication with developers, the modules as architectural elements can be tagged with information about the technologies used for realization. Technologies often realize architectural mechanisms that support quality requirements, as flexibility requirements. Section 4.3.3.2 gives examples of architectural mechanisms, as they are partially also realized in technologies. More details on flexibility mechanisms introduced by SOA are described in Chapter 6. With respect to modules and change impact, technologies are typically seen as black boxes, as they are externally developed and cannot be changed (of course there are exceptions like in-house development of technologies, open source technologies, or individually contracted technologies). Thus, technologies rather realize infrastructure elements or merge into other modules.

Typical architectural views	Typical architectural views providing relevant information for flexibility analyses are development time views with a focus on functions, data, user interface, and processes. Depending on the view-framework used, different names for the views are found: Implementation View [Kru03], Module View [CBB10], Development View [Kru95].
Role of runtime elements	As described above, runtime architectural elements and architectural views play only a minor role in terms of constructing and evaluating a software system for flexibility. However, in practice there are some notable relationships. First, runtime architectural elements are often better related to functional requirements and thus they can be used as entry point for identifying potential change impacts (assuming traceability from runtime architectural elements to development time architectural elements). Second, in practice there is often no clear separation between runtime and development architectural views and architectural elements. Rather, they are mixed or even unified, which denotes a simplification of the architectural model. This can be fully accurate if runtime elements are realized one-to-one by corresponding development time elements. Then, flexibility analysis can work on runtime elements as an approximation of development time elements.

4.3.3.2 Measurement and Metrics of Flexibility

The ability to measure the degree of achievement of a quality attribute is the prerequisite for constructively approaching it. Several approaches towards measuring flexibility, maintainability, or modifiability exist (see Section 3.3). One key contribution of this thesis is to overcome today's situation that evaluating flexibility is an effort-intensive manual task. Rather, we aim at automating the evaluation of the current flexibility level and provide feedback to the architect in nearly real-time. By this, an architect can be supported during architecture design by indicating insufficient degrees of flexibility at an early stage. Then, the architect can revise design decisions for improvement with relatively low effort. This contribution is described by R.D3, R.C3, and R.I3 in Section 1.4.

Measurement goal The major goal of our measurement approach and metrics for flexibility is (formulated according to the GQM (Goal Question Metrics) approach [BD88]):

Analyze the degree of flexibility of a software system with respect to a flexibility requirement with a focus on automatic analysis and real-time feedback about the degree of flexibility from the perspective of a software architect in the context of software architecture design.

Table 5 lists categorized requirements for our flexibility metric, derived from the measurement goal.

<i>Category</i>	<i>Requirements</i>
Measurement subject	<ul style="list-style-type: none"> Alignment with flexibility definition (see Definition 10) Measurement of “true flexibility” (anticipated flexibility requirements matched by flexibility potential) Metric covers “intuitive idea of flexibility”
Measurement results	<ul style="list-style-type: none"> Allow comparing architectural solutions with respect to their flexibility (with respect to a set of flexibility scenarios) Applicability on single flexibility requirements and system parts Hierarchical aggregation of flexibility results up to the overall system and all flexibility requirements
Ease of use	<ul style="list-style-type: none"> Automated calculation of metric results No complicated data as manual input needed No impact on existing architecture models and views
Accuracy	<ul style="list-style-type: none"> Allow working with flexibility scenarios of different levels of accuracy Extensibility of metric for further influence factors Allow computing approximated results with simplified input

Table 5: Requirements for flexibility metric

Measurement idea One main obstacle towards the automated measurement of flexibility in existing approaches is the gap between informally specified requirements and the informal description of architectural decisions. This gap between the problem space and the solution space does not allow the automated analysis of change impacts, as the respective models are lacking information and formality. The key idea (R.I3) for this contribution is that the architect modeling the architecture of a system indicates change impacts of concrete changes (expressed as flexibility scenarios) as part of the architecture model. In order to easily model and represent the information about change impacts in the architecture model, the *Change Im-*

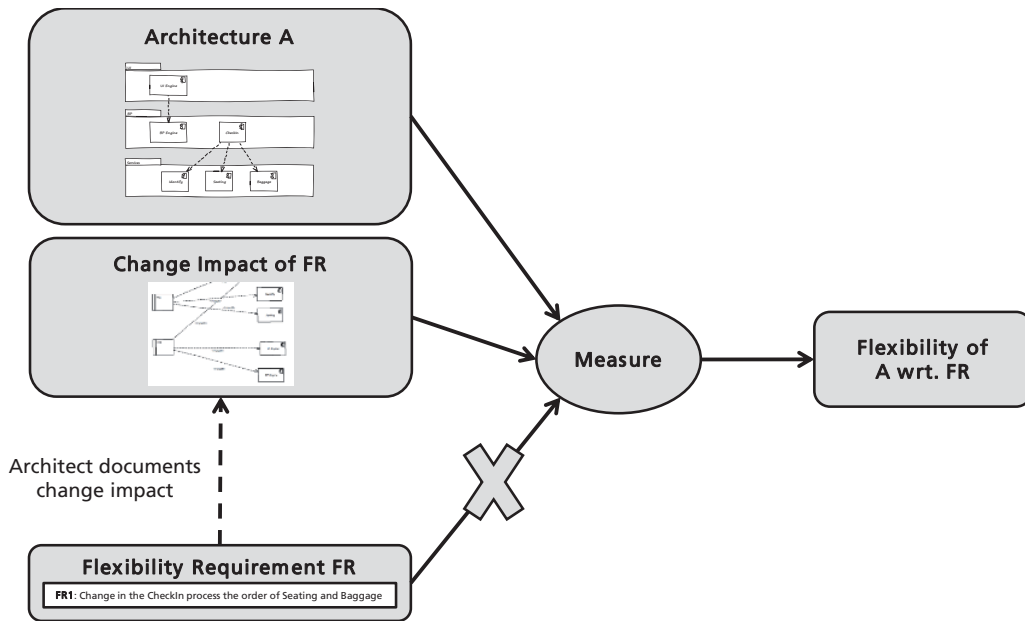


Figure 31: Principle of measuring flexibility

pact View is introduced (see Section 4.3.3.3). The architecture design method incorporating this measurement approach is described in Chapter 5. The resulting measurement model for analyzing the flexibility with respect to one flexibility requirement is depicted in Figure 31.

In the following, we introduce our metrics for flexibility. As flexibility is primarily defined by the effort to change the implementation, we start with the metric definition at implementation level. Architecture is an abstraction of a system’s implementation and it allows earlier (no implementation available) and easier (lower complexity and details) analysis of flexibility. Thus, we approximate the measurement of flexibility at the architecture level. Finally, we describe the aggregation of flexibility results for single flexibility scenarios to an overall flexibility value.

Implementa-
tion level
metrics

At the implementation level, flexibility means minimal impact of changes that are described in flexibility requirements. Thus, we count the changed lines of code (LOC_{changed}) in the overall code-base. Changed means a deleted LOC, a changed LOC, or an added LOC. As the code-base, we see the entirety of development artifacts, as described in Section 4.3.3.1.

Flexibility is a function (FLEX) that calculates for a given implementation ($IMPL \in \mathbb{I}$) of a system and a flexibility requirement ($FR \in \mathbb{F}$) the ratio of change impact to the overall implementation size. To address the requirements of comparability and results aggregation (Table 5), we define flexibility on a $[0, 1]$ interval. 0 denotes low flexibility and high change effort, 1 denotes high flexibility and minimal change effort. In order to normalize the change impact, we define the change ratio (CHR).

$$FLEX: \mathbb{I} \times \mathbb{F} \rightarrow [0, 1]$$

$$CHR(IMPL, FR) = \begin{cases} 1, & \frac{LOC_{changed}}{LOC_{Initial}} > 1 \\ \frac{LOC_{changed}}{LOC_{Initial}}, & else \end{cases}$$

Please note that for the change ratio at implementation level the case could occur that more LOC are changed than LOC are initially there (in case many new ones are added). We cover this in the definition of the function. At architecture level, the definition is slightly changed and this case does not occur.

We define the function FLEX with the curve depicted in Figure 32. It is based on the change ratio, but it defines a tailored function which expresses mainly two aspects.

- We define flexibility to be “1” not only when there is no change at all, but we allow change impact up to a threshold, which is 10 LOC as our default.
- We define flexibility to be “0” not only when the overall implementation is impacted, but already when there is a significant change impact that makes the system hard to change, which is 10% of the overall LOC as our default.
- Between these thresholds, our flexibility function is linear.

In our metrics, we only calculate the change impact in terms of changed LOC in the implementation. This metric is not intended to be used in our method. Rather, it is needed as a preparation for our metric at architecture level, which is defined below and used in our method. We do not cover the calculation of change efforts in terms of time or cost. We assume for simplicity an equal distribution of change difficulty for each type of implementation artifacts (that means, that for example changing 1 LOC of a Java class is equivalent to changing 1 LOC in an XML-based

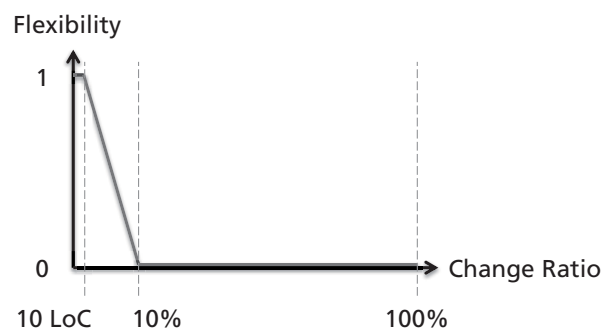


Figure 32: Flexibility metric function definition

process definition language).

Architecture level metrics

Measurement of flexibility at architectural level is essential as it allows flexibility prediction when no implementation is available yet and it can be done at a level of abstraction that allows controlling the complexity of large systems. Thus, we follow the key ideas introduced for measurement of flexibility at implementation level and approximate the flexibility results by measuring at the architecture level.

The key architectural elements to approximate the system size and the change impact are *Modules*. As a system (SYS) is decomposed into modules (MOD) at architectural level, they can be used to estimate their size on a more local level and conclude the overall size of a system. We call modules *atomic* if they do not group further modules.

$$LOC_{Initial}(SYS) = \sum_{i = \text{Atomic Modules}} LOC_{Initial}(MOD_i)$$

In order to estimate change impacts we introduce two concepts: *Impact Type* (IMP_T) and *Impact Size* (IMP_S).

- The key reason to introduce the *Impact Type* concept is that we want to distinguish between changes affecting existing modules or changes requiring the creation of new ones, or changes requiring the deletion of existing ones. Modules that are newly created are not counted as impacting the existing implementation. Only the changes of existing modules to appropriately include the new modules are counted. Thus, we introduce the basic operations *Add Module*, *Modify Module*, *Delete Module* for modules (also used in [SR09, VEG08]). Adding a module means at implementation level to create new source code artifacts. Modifying a module means at implementation level to make changes inside existing source code artifacts. Deleting a module means at implementation level to remove an existing source code artifact.
- The key reason to introduce the *Impact Size* concept is that we want to ease the estimation of change impacts for architects. Thus, an architect does not have to exactly estimate the number of lines of code affected, but rather ranks a change on a scale [*low, medium, high*]. The impact size is only relevant for the impact type *Modify Module*. The impact size values are then translated into a change ratio for the module, the defaults are *low=0.1, medium=0.3, high=0.5*.

In the following, we define the approximation of impacted lines of code in a module for a flexibility requirement. Then, the impacted lines of code for the overall system can be calculated.

$$LOC_{Changed}(MOD, FR) = \begin{cases} LOC_{Initial}(MOD) * IMP_S(MOD, FR), & IMP_T(MOD, FR) = Modify \\ 0, & IMP_T(MOD, FR) = Add \\ 0, & IMP_T(MOD, FR) = Delete \end{cases}$$

$$LOC_{Changed}(SYS, FR) = \sum_{i=Atomic\ Modules} LOC_{Changed}(MOD_i, FR)$$

Then, we can calculate the change ratio of a flexibility requirement on a system:

$$CHR(SYS, FR) = \frac{LOC_{Changed}(SYS, FR)}{LOC_{Initial}(SYS)}$$

Finally, the flexibility $FLEX(SYS, FR)$ is determined according to Figure 32, analogous to the implementation level. Further refinements of the flexibility metric are possible. For example, it is meaningful to decrease the flexibility value in case of strong scattering of changes: Many different locations to change typically cause more effort than one local but larger change.

Using modules as a hierarchical decomposition of a system at architectural level has several benefits for the calculation of our flexibility metric. First, it allows easier estimation of implementation sizes by limiting the scope. Second, it allows having modules with different levels of granularity for different parts of the system. Third, it allows local and incremental refinement of size measures. Forth, it allows connecting the module size to the implementation size of actual source-code in case an implementation exists.

As described in Sections 4.1.3 and 4.3.3.1, the estimation of change impacts always has to be done at the modules that are manually changed or created.

Aggregation
to overall
flexibility

So far, flexibility metrics are defined for single flexibility requirements only. Now, we aggregate the flexibility values to a flexibility result of a system with respect to the entirety of flexibility requirements.

$$FR_{SYS} = \{FR \mid FR \text{ is anticipated flexibility requirement for } SYS\}$$

As described in Section 4.2.2, flexibility requirements can differ in their probability, priority, or frequency. Thus, we introduce a weighting for the calculation of the overall flexibility which can include these factors. w_i is the weighting factor for FR_i .

$$FLEX(SYS, FR_{SYS}) = \frac{1}{\sum w_i} \sum_{FR_i \in FR_{SYS}} w_i * FLEX(SYS, FR_i)$$

Discussion Our flexibility metric is intentionally constructed with a focus on implementation effort and change impact. This makes the metric more comprehensible, in particular compared to complex mathematical metrics, like the maintainability index [CAL+94]. The flexibility metric as defined at architecture level is used in our tool extension (see Section 5.4) in order to give quick feedback of achieved flexibility to architects during architecture construction or rework. All the influence factors introduced for the metric are incorporated in the tool. Thresholds like in the flexibility definition (see Figure 32) or the *Impact Size* factors can be configured in the tool. In the following section, the *Change Impact View* is described in order to allow architects easy modeling and entering of information necessary for calculating flexibility.

4.3.3.3 Change Impact View for Automated Flexibility Analysis

In Section 4.3.3.2, we introduced the idea of how to measure flexibility with a high degree of automation. The key aspect of this measurement is that the architect makes his reasoning about the change impact of flexibility scenarios explicit while designing the architecture. Additionally, we described the different factors we include in our flexibility metric. One requirement from a method perspective is that the information an architect has to provide can be easily entered (see Table 5), without disturbing the architecture design process but rather supporting it.

In order to be able to run automated analyses on the architecture, all information about the change impact of flexibility requirements have to become part of the architecture model, too. Thus, we extend the architecture model by the *Change Impact View*. In this view, all the information necessary for calculating the flexibility metric can be easily modeled by an architect or entered as attributes. The meta-model of the change impact view is depicted in Figure 33. *Flexibility Scenario* becomes a first-class element of the architecture model. In many architecture modeling approaches (as in ACES-ADF), architectural requirements are already part of the architecture model. An *Architectural Element* can be any element of the architecture that is considered an implementation artifact which is impacted by the change described in the flexibility scenario. Foremost, architectural elements in the change impact view are *Mod-*

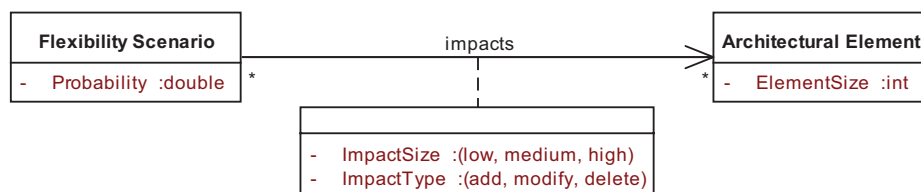


Figure 33: Meta-model for change impact

ules, but depending on the way of modeling (see Section 4.3.3.1) they can also be interfaces, components, connectors, data elements, etc. The impact of flexibility scenarios on architectural elements is explicitly modeled with an *Impacts*-relationship. Beyond the model elements, our flexibility metric requires some more detailed information (*Probability* of occurrence of flexibility scenario, *ImpactSize* and *ImpactType* of the impacts-relationship, *ElementSize* in terms of lines of code for architectural elements). The architect can simply enter or select the appropriate information as tagged values of the model elements. Covering this model information, an architectural modeling tool can calculate the flexibility metrics as defined above and provide the results to the architect (see Section 5.4).

4.3.4 Cost Considerations of Flexibility

So far, we have mainly discussed the benefits of a software system offering flexibility. However, flexibility always comes at a price. First, flexibility requirements might be competing with other requirements (see Section 4.2.3). Second, building in flexibility can mean to increase the complexity of a software system, which might increase the effort for particular maintenance tasks. Third, flexibility is often achieved by additional architectural mechanisms and indirections which cause additional complexity and effort in the implementation of the system.

In particular in the light of cost considerations, knowing probabilities and priorities is important when designing for flexibility. Architects can discuss with the relevant stakeholders about the level of flexibility to realize which is an investment and leads to potential benefits during later changes. In [Bah05], a detailed discussion of cost aspects is described. The key idea is to apply concepts of Real Options Theory to flexibility considerations (see Section 3.3.4). In practice, a quantification of all involved cost factors related to flexibility is often impossible. Thus, an architect has to take the key cost factors into account and make justifiable decisions.

4.4 Conceptual Model of Flexibility

In the previous sections, we characterized flexibility in detail, covering all aspects from flexibility requirements over architecture as a means of construction and evaluation of flexibility down to the implementation which is in the end impacted by changes to a system. In this section, we summarize all these aspects in a conceptual model. Thus, we do not explain each concept and each relationship in detail again, but we focus on giving a brief overview and appropriate references to the detailed explanations above.

Purpose of the model	The conceptual model, together with its details described before, provides a formalization of flexibility as a quality attribute of software. It serves as the foundation for the methodical contributions presented in Chapter 5. The conceptual model provides a level of formalization which is aligned with the needs of the methodical ideas. Thus, the measurement of flexibility is fully formalized as it is intended to be conducted with tool-based automation support. The methodical guidance for the construction of flexible systems is less formalized: There, the target is mainly to put all concepts involved in flexibility in relation to each other as support for architects.
Target audience	For all technical roles involved in software development, the conceptual model is intended to be a guidance for understanding what flexibility is and what it means to deal with flexibility in software development, both constructively and analytically. The conceptual model is intentionally separated in different views. This allows concentrating on particular aspects of the model. Additionally, model views with a limited number of model elements and limited complexity can better serve as mental models technical stakeholders can remember and use.
Model views	<p>The overall conceptual model is represented in four model views. Figure 34 depicts these views and the conceptual areas they mainly cover. The conceptual areas are organized along the artifacts and activities in software development: First, there are flexibility <i>Requirements</i>. Second, there is <i>Architecture</i> as the set of architectural decisions facilitating flexibility and as the means to analyze flexibility at an appropriate level. Third, there is the <i>Implementation</i> which typically causes the main effort for changes. This effort has to be minimized by flexibility. Finally, <i>Measurement</i> is also a conceptual area of our model since measurement of flexibility with a high degree of formalization is important for our methodical support with architecture-tool-based flexibility measurement. The conceptual model comprises the following model views:</p> <ul style="list-style-type: none"> • <i>Flexibility Core View</i>: Flexibility requirements and how to address them at architectural level. • <i>Architecture Construction View</i>: Architectural aspects in detail, in par-

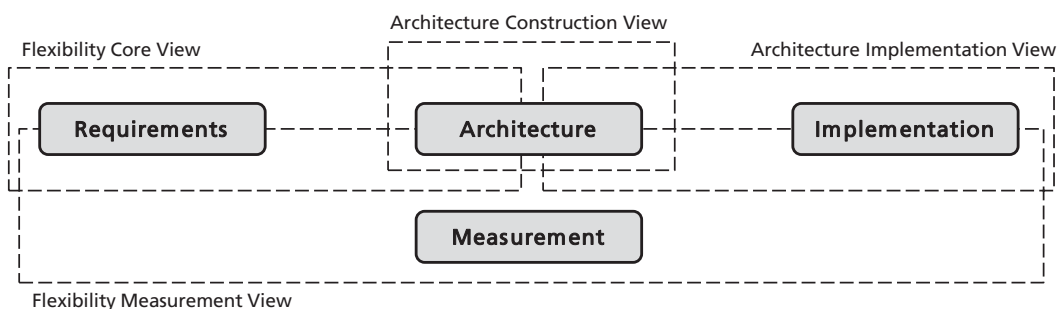


Figure 34: Views of the conceptual model for flexibility

ticular the concepts introduced for constructively approaching flexibility.

- *Architecture Implementation View*: Relationships between architecture and an implementation of a system with respect to flexibility.
- *Flexibility Measurement View*: The relevant aspects across flexibility requirements, architecture, and implementation with respect to measuring flexibility.

We explain the model views in a rather brief style, as all the details are described in earlier sections. Thus, we summarize the key ideas and link to the sections that cover the relevant aspects.

Flexibility Core View

The flexibility core view (see Figure 35) relates the most important concepts around flexibility. The distinction between *Flexibility Requirements* and *Flexibility Potential* is introduced and the match of both is called *True Flexibility* (Section 1.3). *Flexibility Requirements* express the potential need for *Changes* in the future (Section 4.2). *Flexibility Potential* is achieved by making adequate *Architectural Decisions* (Sections 4.1.3, 4.3.1). Important *Architectural Decisions* are about *Architecture Mechanisms* (Section 4.3.1.2) and *Business Logic Mapping* (Section 4.3.1.3); that is how a system’s functionality is mapped to architectural elements.

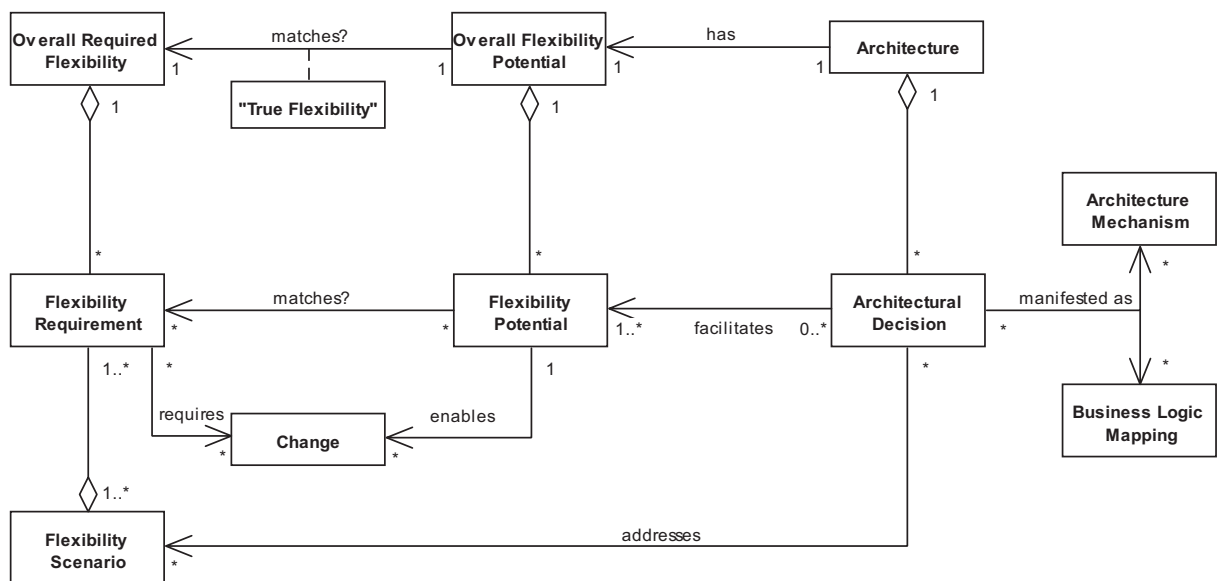


Figure 35: Conceptual model: flexibility core view

Architecture Construction View

The architecture construction view (see Figure 36) aims at explaining and relating the most important concepts needed for constructively addressing flexibility at architecture level. This view is compatible and aligned with typical architecture meta-models, as for example ACES-ADF [KKN11] or the IEEE Recommended Practice for Architectural Description [IEEE00]. However, it focuses on and extends typical architectural models with respect to flexibility.

Architecture consists of *Architectural Decisions* and *Architectural Elements*. Important *Architectural Decisions* are about *Architecture Mechanisms* (Section 4.3.1.2) and *Business Logic Mapping* (Section 4.3.1.3); that is how a system’s functionality is mapped to architectural elements. *Architectural Decisions* aiming at flexibility are guided by general Architecture Principles (Section 4.3.1.1). *Architectural Elements* are grouped into *RunTime Elements* representing entities in a running system, and *DevTime Elements* representing entities in a software development process (Section 4.3.1, 4.3.3.1). *Modules* are the most important *DevTime Elements* from a flexibility perspective (Section 4.3.3.1). *Architectural Elements* are organized in *RunTime Views* and *DevTime Views* respectively (Section 4.3.3.1). For the construction and documentation process of flexible architectures, we introduce the distinction in *Infrastructure Ele-*

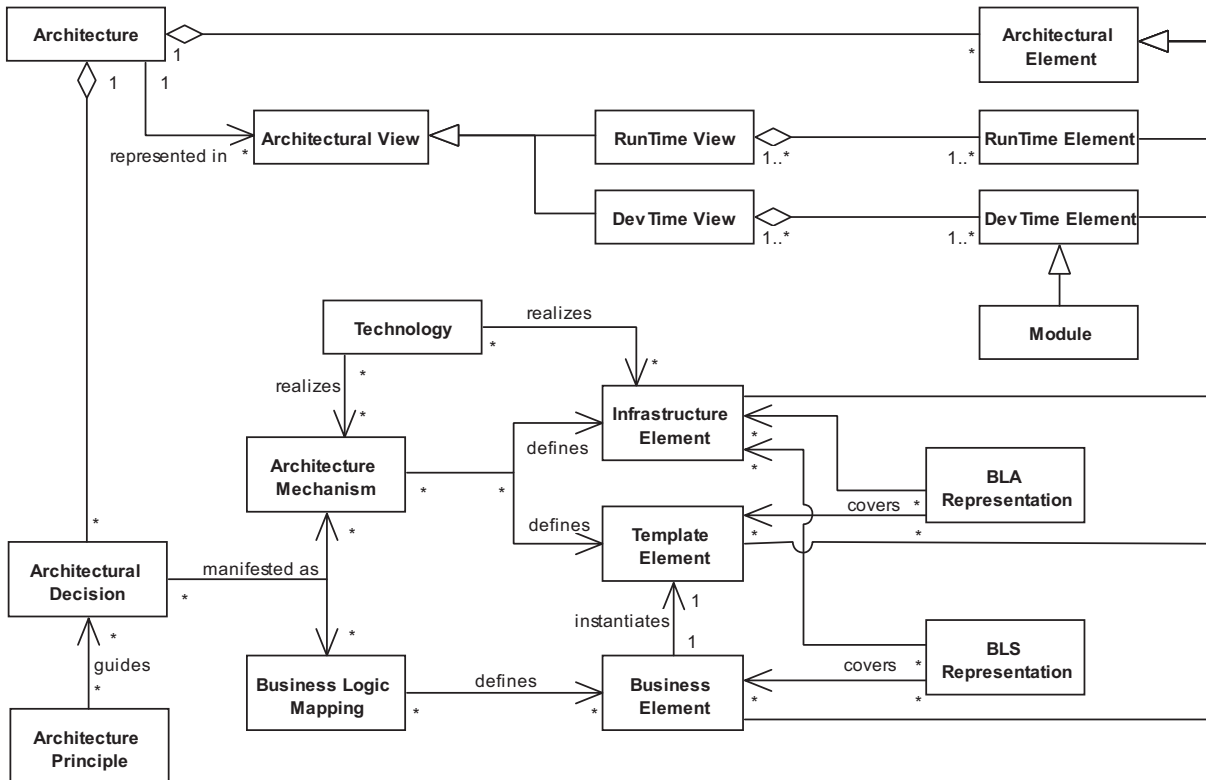


Figure 36: Conceptual model: architecture construction view

ments, *Template Elements*, and *Business Elements*, which are *Architectural Elements* (Section 4.3.1.3). We distinguish two levels of flexibility-relevant architecture: *Business Logic Agnostic* (BLA), and *Business Logic Specific* (BLS). Whereas *BLA* focuses on the *Architecture Mechanisms* only, *BLS* also covers the *Business Logic Mapping*. Technologies are important for architecture as well. They realize *Architecture Mechanisms* and provide ready-to-use implementations of *Infrastructure Elements* (Section 4.2.2, 4.3.3).

Architecture Implementation View

The architecture implementation view (see Figure 37) aims at describing the relationship between architecture and implementation of a system with a focus on flexibility.

A *Software System* has an *Implementation* as its main constituting part. The *Implementation* consists of a number of *Implementation Artifacts* which can be source code files, descriptors, models that are used in model-driven technologies, etc. The *Implementation Artifacts* are important for flexibility, as the main effort for conducting changes typically has to be spent on changing these artifacts (Sections 4.1.3, 4.3.1). Architecture is the level of abstraction on which to reason about the impact of changes in complex systems. *Architecture* provides an abstraction of the *Implementation* and on a more detailed level, *DevTime Elements* provide an abstraction from *Implementation Artifacts*. With the help of this abstraction, early predictions about change impact as well as appropriate *Architectural Decisions* can be made (Section 4.3.1).

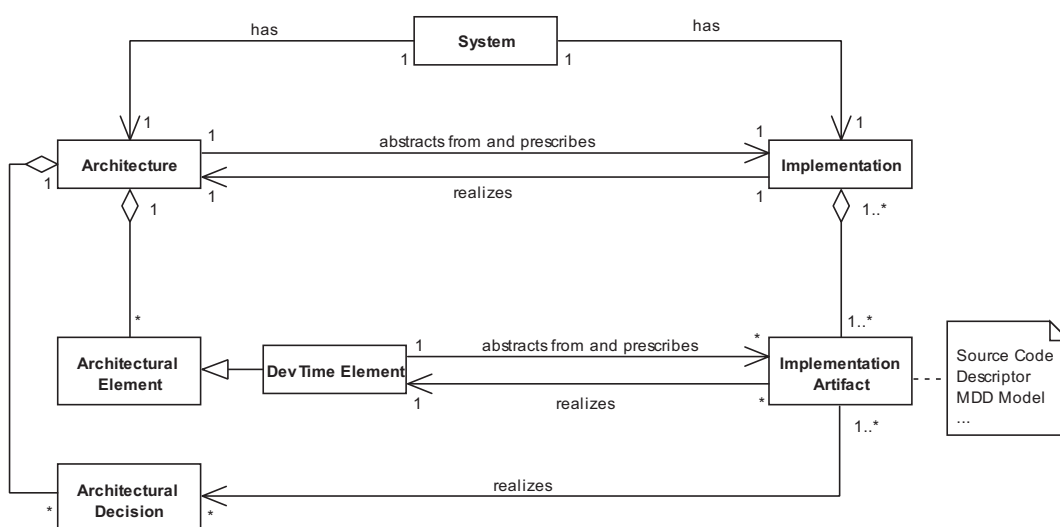


Figure 37: Conceptual model: architecture implementation view

Flexibility Measurement View

The flexibility measurement view (see Figure 38) covers all aspects of the model that are relevant for measuring flexibility according to our *Flexibility Metric* (Section 4.3.3.2). A key element for measuring flexibility is *Change*. *Change* of a software system is what is described in *Flexibility Requirements* (Section 4.1.1). If a system is flexible with respect to a particular *Change*, it allows this *Change* to be conducted with little effort, which means that it has only a small *Change Impact* on *Implementation* (Section 4.1.1). With our *Flexibility Metric*, we measure the *Change Impact* that a *Change* has on the *Implementation* (Section 4.3.3.2). Although the measurement of *Change Impact* is defined at the *Implementation* level in theory, this is in practice often impossible, due to the complexity of the *Implementation* or due to the fact that the *Implementation* is not available yet (Section 4.3.3.2). Thus, *Architecture* is used as the foundation for an approximation of the *Flexibility Metric* (Section 4.3.3.2). *Architectural Elements* that abstract from *Implementation Artifacts* are used to analyze the *Change Impact* of *Changes*. In order to make modeling the *Change Impact* easier for an architect, *Change Operations* (add, modify, delete) can be used to characterize the *Change Impact* (Section 4.3.3.2, 4.3.3.3). For a more detailed calculation of the

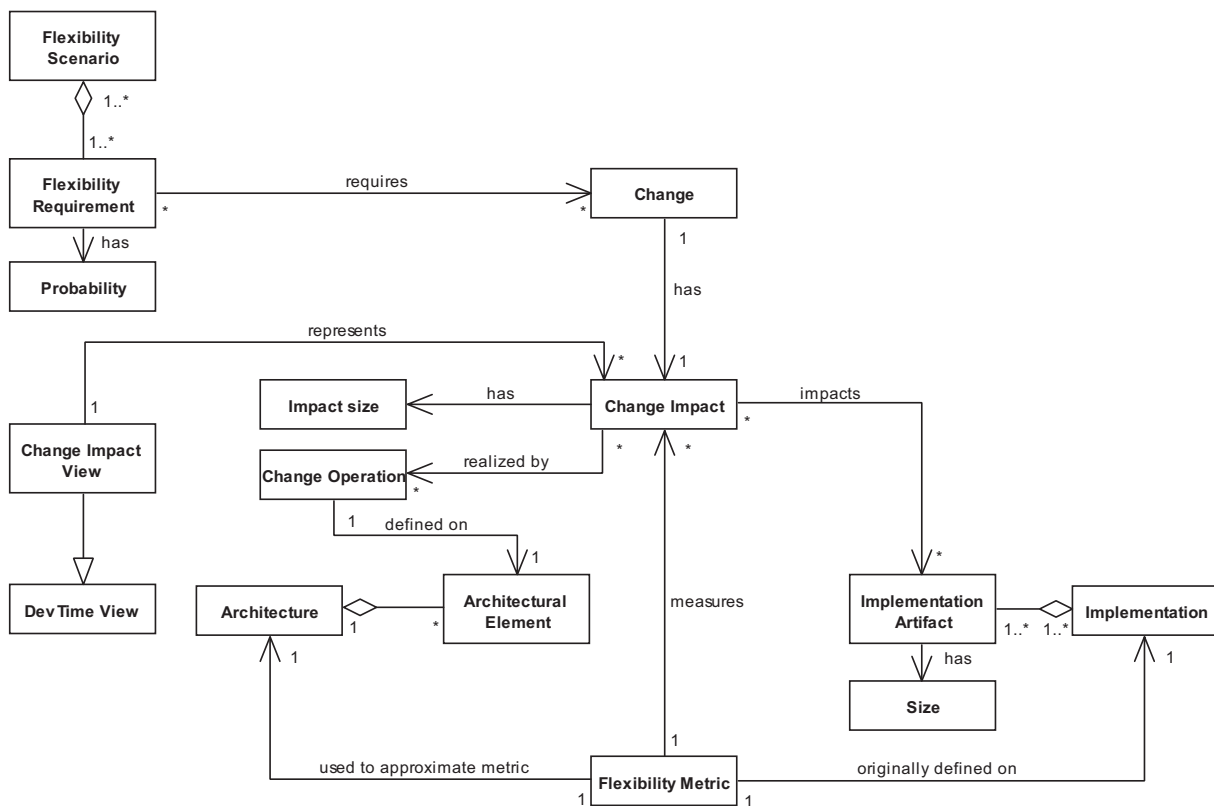


Figure 38: Conceptual model: flexibility measurement view

Flexibility Metric, the Probability of Flexibility Requirements, the Impact Size of Change Impacts, and the Size of Implementation Artifacts can be included (Section 4.3.3.2). The *Change Impact View* is an *Architectural View*, a *DevTime View*, which is used for graphical modeling and representation of *Change Impacts* (Section 4.3.3.3).

In the introduction of Chapter 4, we referenced the following questions about flexibility which were raised in [SHN01]. Chapter 4 addresses most of these questions; the question about *design for flexibility* will be addressed in Chapter 5.

- “What is flexibility? How does a formal definition look like?”
- “Why or when is flexibility needed in system design?”
- “How can one design for flexibility? What are the design principles?”
- “What are tradeoffs associated with flexibility?”

5 Engineering Flexible Software Systems

*“Information technology and business
are becoming inextricably interwoven.
I don't think anybody can talk meaningfully
about one without the talking about the other.”*
Bill Gates

In Chapter 4, we elaborated a conceptual foundation for the quality attribute flexibility. A focus of this foundation is the role of architecture for flexibility and what that means in terms of architectural artifacts. In this chapter, the focus is on how architects can work to design flexible systems. Therefore, we describe engineering activities at architectural level which aim at achieving adequate flexibility.

Explicitly designing for flexibility means to make architectural decisions that allow conducting anticipated changes with minimal effort. Besides this decision making, also the evaluation of architectural decisions is critical in order to confirm that the architecture design is on the right track, with respect to flexibility and other requirements. These two activities, constructive guidance for decision making and continuous evaluation of design decisions, are the key activities supported by methodical contributions of this chapter.

We start with a methodical overview on all activities involved and on their interplay in Section 5.1. Then, the key activities are explained in detail. We begin with a brief explanation of the elicitation of flexibility scenarios in Section 5.2. After that, the explicit guidance of architects during architecture design is introduced in Section 5.3. Continuous measuring of flexibility in the context of architecture tools is described in Section 5.4. Finally, we close the chapter with a discussion of the engineering support for flexibility in Section 5.5.

5.1 Methodical Overview

In the context of software development, architecting plays an important role. At the architectural level, the key design decisions for the fulfillment of functional requirements and quality requirements like flexibility have

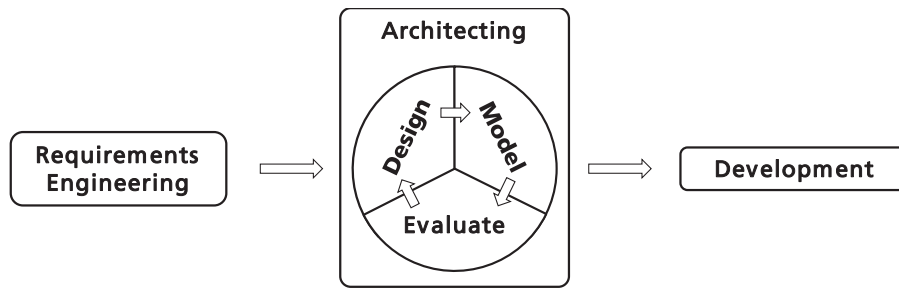


Figure 39: Architecting as activity between requirements engineering and development

to be adequately made. It is important to get these key decisions right at the architectural level as they are implemented afterwards in a distributed manner by different teams, which makes rework very time-consuming and expensive.

Getting the key architectural design decisions right basically means two things: First, decisions have to be made, which is a constructive process of deriving solutions from the given requirements. Second, based on the decisions made (manifested in the architecture), a prediction has to be made whether this architecture is adequate for fulfilling the requirements at hand (see Sections 2.1.3 and 4.3.2). Such a prediction is typically done in an evaluation of the architecture, using different methods depending on the quality attribute at hand and the level of accuracy needed. This is true for any type of quality attributes, and thus also for flexibility. If the prediction finds that the flexibility needed is not achieved, rework of the architecture is needed. The resulting architecture is the input for subsequent development activities.

Figure 39 shows architecting as an activity embedded between requirements engineering and development. The elicitation of flexibility requirements and their documentation as flexibility scenarios is an important task for requirements engineering. It is briefly sketched in Section 5.2, but the main focus of our methodical contribution is on architecting.

Key architecting activities

As described above, Designing (in the sense of making decisions) and Analyzing (in the sense of checking adequacy of decisions) are key activities of Architecting. This is also depicted in Figure 39. Additionally, Modeling is a key activity which denotes documenting the decisions made in a form that is well usable for further needs during architecting and other development activities. These three activities are conducted in an iterative and incremental way. These highest-level activities are more or less part of all architecture design methods. In [HKN+07], five such methods are surveyed and compared and similar sequence of activities is described there. There, Design and Modeling are put together. Additionally, there is an initial analyzing step which in our case is part of requirements engineering and Design. The key point here is that our methodical

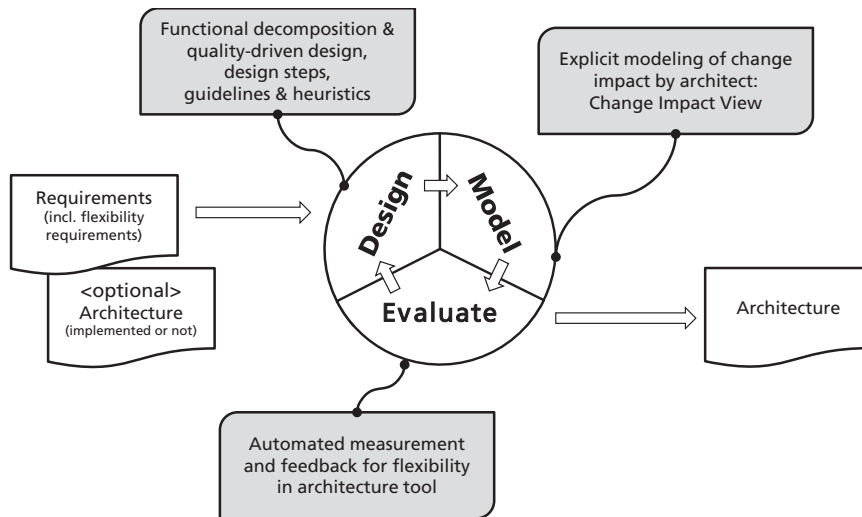


Figure 40: Contributions to the architecting activities

contributions for flexibility are not tailored to a specific architecting method, rather they are universally applicable to other methods.

Methodical contributions

In Figure 40, our methodical contributions for achieving high flexibility are annotated to the architecting activities. First, we give guidance for the design of architectures with a focus on flexibility (Section 5.3). The design process is supported by intertwining the aspects of functional decomposition and quality-driven design. Guidelines and heuristics for making appropriate decisions for flexibility are sketched. Second, we provide support for the evaluation of achieved flexibility (Section 5.4). It is the goal to automate the evaluation of flexibility as much as possible in order to give continuous feedback about the level of flexibility to the architect. To achieve this, we add in the modeling step the explicit modeling of change impact, which is done in the change impact view (see Section 4.3.3.3). Then, we can offer automated measurement of flexibility as part of architecture modeling tools in the evaluation step, which allows shortening the evaluation cycles and thus allows a faster convergence to an adequate architectural solution. The methodical contributions described are mainly targeted at architects.

Relationship to other architecture methods

Our methodical support for architects can be applied with typical architecture design methods. It is not intended to replace any existing architecture method; rather it is complementary and builds on existing methods. Visually speaking, our contributions can be seen as a conceptual *PlugIn* for architecture methods which handles the specific concern *Flexibility*. We also emphasize this interpretation by the visualization in Figure 40, where we show the single contributions in a typical architecture design method. In the case of the methodical guidance, the *PlugIn* idea is to be understood rather conceptually; in the case of the flexibility measurement, it is really realized as a *PlugIn* for the architecting tool Enterprise Architect (called *AddIn*) (Section 5.4.3).

In order to achieve this goal of supporting flexibility through all architecting activities, we thoroughly analyzed flexibility as a quality attribute of software in Chapter 4. With the help of this knowledge, the guidance in the single activities of architecting can be made more concrete than they are when targeting at design for all types of quality attributes.

Eventually, it is the goal of architecting support that all relevant quality attributes are covered in a similar manner. For selected quality attributes like performance, such approaches already exist (e.g. [BKR09]). To be most effective, these different quality attribute “PlugIns” should work on a common architecture meta-model and in an integrated tool platform. Then, analyses about the mutual influence of design decisions for the one or the other quality attribute can be conducted.

Contexts of
method
application

Architects can apply the method for achieving flexibility in different contexts. The context of designing a new software system is the most obvious one. However, it can also be applied in the context of software migrations or other evolution activities. As the requirements for a software system in general change over time, this can also be true with respect to flexibility requirements. That is, architects discover new flexibility requirements, for example when similar changes occur in a recurring way which are not supported by adequate flexibility mechanisms and which are thus overly costly. Then, such a change project can also be used to redesign the architecture for better flexibility. Additionally, larger evolution projects conducting massive changes to a software system can be used to introduce new flexibility potential if needed. The change impact view as an instrument for automated measurement of flexibility can also be applied in other contexts than architecture design. Flexibility evaluations might be necessary for different reasons like for checking how future-proof a software system is. Such evaluations can also be supported, which requires to model the change impacts manually in the change impact view. This then allows automated analyses of flexibility considering particular changes.

5.2 Eliciting Flexibility Scenarios

Anticipation
of flexibility
requirements

When change requests to a system arise, they should be covered by the flexibility potential of the system; otherwise conducting the change is effort- and cost-intensive. Achieving this coverage means not only to make the right architectural decisions but also to anticipate the needed flexibility requirements during system design.

This thesis strongly focuses on the architecture design for flexibility. However, we give a brief introduction to requirements engineering for flexibility in this section.

Flexibility requirements are particularly difficult to elicit as they always deal with anticipated changes in the future which are differently likely to happen. Further, there might be context factors changing around a software system which were not even considered. Thus, a hundred percent coverage of flexibility requirements can hardly be achieved. However, experience of other software systems and a strong domain knowledge help to come up with the right flexibility requirements for a software system. The goal to be followed for flexibility requirements is to make a sound trend analysis and to approximate future changes as well as possible.

Using existing requirements methods

We do not contribute a new methodology for eliciting flexibility requirements in this thesis. Rather, we rely on existing methods for requirements elicitation and add information from our characterization of flexibility to make these methods more effective. In [Doe11], the NFR Method is described, which explicitly aims at eliciting non-functional requirements. It deals with quality models describing quality attributes which are used to support requirements engineers in the systematic elicitation of non-functional requirements. Our characterization and classification of flexibility scenarios (see Section 4.2.2) could be used in the context of this method.

Another method which can be adopted for eliciting flexibility requirements is PuLSE-Evo (Product Line Software Engineering – Evolution) [VEG08]. It was originally developed to support the evolution of software product lines. In this context, the elicitation of future needs and potential changes is also relevant. Although in the first place the method was developed for embedded systems, it is mainly transferrable to other system types like information systems. PuLSE-Evo has an own conceptual model of software evolution which partially overlaps with our conceptual model of flexibility. These models have a different focus and partially slightly different terminology, but in general they are compatible (e.g. Impact in the PuLSE-Evo model means the impact of a change on the context in terms of business aspects, whereas Change Impact in the flexibility model means the impact of a certain change on architectural or implementation elements). A concrete mapping is possible, but not in the scope of this thesis. For this mapping, again our characterization and classification of flexibility scenarios (see Section 4.2.2) would be helpful.

Flexibility scenarios

After eliciting and characterizing the flexibility requirements for a system, they should be represented as flexibility scenarios (see Section 4.2.1). It is not always possible to anticipate flexibility requirements for a system. This might be due to inexperienced stakeholders or to a largely new domain addressed. Then, it might be beneficial to define, based on experience of the architect, some “standard flexibility scenarios” which cover more technically motivated changes that often occur.

In the next section, we describe the enhancement of architecture design methods with better guidance for flexibility, making appropriate decisions to address the flexibility scenarios.

5.3 Architecture Design for Flexibility

In this section, we describe what we contribute to architecture design methods in order to better support achieving flexibility. Architects get concrete guidance for making architectural decisions related to flexibility. This guidance is mainly given by splitting otherwise complex design aspects and by giving heuristics and guidelines.

We start with a definition of design goals for our methodical support (Section 5.3.1). Then, we give an overview of the design process as supported (Section 5.3.2). Finally, we describe in detail the individual activities of the design process with heuristics (Section 5.3.3).

In Figure 40, we conceptually distinguished Design and Modeling. With respect to flexibility measurement, the explicit Modeling activity is important. However, for all design activities in this section, design and modeling is closely interwoven and modeling is not explicitly described. Rather it is assumed that the resulting architectural decisions and structures are modeled as they are designed.

5.3.1 Design Goals

Key goal: true flexibility	The foremost design goal with respect to flexibility is to achieve <i>True Flexibility</i> . According to Definition 7 (see also Figure 6), this means that flexibility requirements are appropriately covered by the built-in flexibility potential. This bases on the assumption that the anticipated flexibility requirements approximate the eventually occurring change requests adequately. Additionally, not too much flexibility potential should be built in, as flexibility potential always comes at a cost (see Section 4.3.4). Constructively achieving <i>True Flexibility</i> means to make the adequate architectural decisions.
Minimize change im- pact	In Section 4.3.1, we analyzed what makes an architecture flexible and how flexibility potential can be achieved. The key means to achieve flexibility is (according to the definition of flexibility) to <i>minimize and localize the change impact</i> . This minimizes the effort for changes in the sense that only a small number of implementation elements have to be touched and the key architectural decisions stay stable.
Reduction of complexity	Designing for flexibility is challenging as it involves considering many different aspects like functional decomposition and flexibility mecha-

nisms, business logic and technologies, the interplay between runtime and development time aspects, etc. Making this inherent complexity better manageable by explicit addressing and partial separation in the architecture design process is our leverage towards higher flexibility.

Context coverage Our methodical support should address the relevant contexts, as described in Section 5.1. That means in particular that it should be applicable to architecting of new systems as well as to the evolution of existing systems.

5.3.2 Design Process Overview

Context In this section, the overview on the design process is described. According to Figure 40, the design process covers mainly the *Design* activity and implicitly the *Modeling* activity for the architecture. The design progress addresses the flexibility related aspects as indicated in Figure 40. In the overview, we describe the key ideas of the design process and of how the single activities belong together as well as what the overall sequences of activities look like. In the next section, we describe the details of the single activities.

Contribution and principles The key contribution of the design process is to reduce the inherent complexity of designing for flexibility by making activities, artifacts, and the relationships more explicit where possible. Additionally, we support the activities with flexibility-specific heuristics and guidelines helping the architect to master the complexity. Where typical architecture methods stay rather abstract in order to be able to address all kinds of requirements, our approach focuses on flexibility and gives the accordant guidance. The key principles to realize this are the following:

- *Consider functional decomposition (incl. business logic mapping) and quality-driven design (incl. application of flexibility mechanisms):* Both, the selection of appropriate flexibility mechanisms and the appropriate mapping of business logic are relevant for achieving flexibility (see Section 1.3).
- *Consider business logic and technology aspects:* Flexibility is often supported by using technologies which realize certain flexibility mechanisms. These technologies have to be appropriately combined with the business logic of the system (see Section 4.4)
- *Consider runtime aspects and devtime aspects of a system:* Flexibility is a devtime quality attribute. However, during system design there is a close relationship to runtime quality attributes. This results from the mutual influence of design decisions made for devtime and runtime.
- *Consider top-down decomposition for new requirements and bottom-up analysis of existing realizations:* In case of an existing system, the implementation cannot be changed easily in order to comply with design decisions made for achieving flexibility. Thus, the existing de-

sign decisions have to be taken into account when designing for new flexibility requirements.

Despite the more detailed guidance of the design process and its activities, architecture design is still a task requiring a good portion of experience and creativity.

Input The architecture design process works like typical architecture methods on *requirements* as input. This covers all types of requirements like business goals, functional requirements, quality requirements, or any type of constraints. The key aspect for designing a flexible architecture is that adequate flexibility requirements in form of scenarios are available. Optionally, the design process can also take *existing architectures* into account. An architecture might be already existent when an existing system is evolved or when a new architecture has been designed, but the flexibility of it should be improved in another design iteration. An existing architecture might be available in a well-documented form, but it might be also available in the source code of the existing system, requiring a systematic abstraction and analysis (reverse engineering).

Output The architecture design process produces as output an architecture. The architecture consists of architectural decisions and architectural elements, relationships among them, and further characterizing attributes (see Definition 2 or [BCK03, RH06, TMD09]). An architecture is typically represented using architectural views. The specific aspects of modeling *Change Impact Views* (see Section 4.3.3.3) are covered in Section 5.4.

Process overview Figure 41 depicts the overview on the design process, with the key inputs and outputs, and with the key activities of the process and their flow. These key activities are widely not new but can be partially found in today's architecture design processes, as for example described in [HKN+07]. Our key contributions are as described above.

As can be seen from Figure 41, there is no strict order of activities which would lead to the desired architecture. Rather, the sketched activities might have to be revisited iteratively until the architecture converges to the desired state of fulfilling the requirements. Nevertheless, there are some typical patterns of stepping through the activities:

- Design typically starts with a coarse-grained functional decomposition
- Runtime aspects are typically addressed first, devtime aspects are typically addressed later
- The activities of functional decomposition and realizing quality attributes are intertwined in the sense that decisions are continuously refined
- The realization of runtime quality attributes and devtime quality attributes is dependent on each other and needs iterative refinement

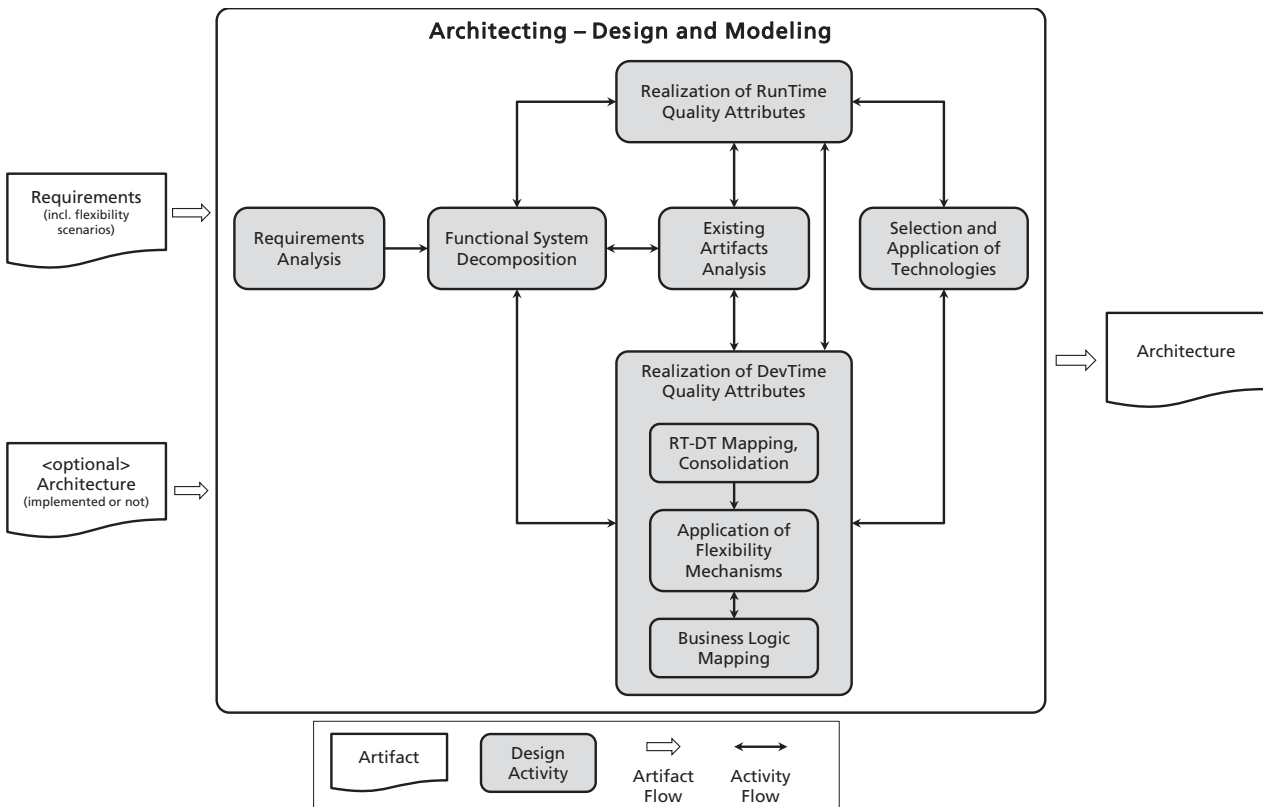


Figure 41: Architecting design process overview

- Technologies are incorporated to realize architectural mechanisms at runtime and devtime

As sketched, the design process is highly iterative and incremental. An appropriate unit to control the iterations are architecture scenarios in general and flexibility scenarios in particular. Intentionally, there are no clear artifact flows sketched between the activities. Rather, the activities work together on the architecture model. Particular activities have a focus on specific architectural artifacts, which is described in detail in Section 5.3.3.

Guidance of the process

Further guidance to the architecture design process can be derived from the conceptual model for flexibility (see Section 4.4) and the Architecture Decomposition Framework (ACES-ADF) (see Section 2.1.3). The conceptual model guides with more detailed information about flexibility and in particular with the role of architecture in achieving flexibility. ACES-ADF guides with the separation of runtime and devtime dimension, and it guides with the separation of aspects like functions, data, processes, UI, and technologies. Concretely, this is described for the single activities in Section 5.3.3.

5.3.3 Design Process Activities

Description of activities In this section, all activities of the architecture design process, as introduced in the previous section, are described in detail. Each activity is described according to the following uniform structure, always focusing on flexibility (other aspects that are generally done or related to other quality attributes are left out).

- Introduction
- Goals for flexibility
- Artifacts consumed and produced
- Guidelines and heuristics for flexibility

An architect applies these activities according to the overall process as described in Figure 41. He incrementally steps through the activities addressing flexibility requirements and making architectural decisions until an adequate level of flexibility is achieved. Please note that these activities typically do not aim at flexibility only. Rather, they are conducted to fulfill all kinds of requirements of a system. Our activity description of the activities contributes to the flexibility-related guidance and does not describe other steps typically conducted in these activities.

Activity: **Requirements Analysis**

Introduction

Requirements analysis is an architectural activity aiming at understanding and processing architecturally-relevant requirements as input for the architecture design activities.

Goals for flexibility

- Identify adequate set of flexibility scenarios to be addressed in overall design and in single design iterations

Artifacts consumed and produced

- Consumed: Flexibility scenarios, other requirements
- Produced: Selection and grouping of flexibility scenarios

Guidelines and heuristics for flexibility

- Select a number of flexibility scenarios for addressing in the overall design and for the next design iteration according to the prioritization and probability of flexibility scenarios

- Group flexibility scenarios according to what is changing or according to the impacted architectural elements (see Section 4.2.2) as units of architectural work in the design process
- Identify functional requirements that are closely related to the business logic covered in flexibility scenarios, in order to better understand the context of potential changes

Activity: Functional System Decomposition*Introduction*

The functional decomposition of software systems aims at decomposing functional requirements in a way that they can be assigned to software elements in the broadest sense. That means, it has to be defined which runtime elements exist and how they interact to deliver the required functionality. Further, it has to be defined which devtime elements exist and how they are organized for development activities.

Goals for flexibility

- Decompose the functionalities in a way that allows later-on mapping of functionality to element types defined by flexibility mechanisms

Artifacts consumed and produced

- Consumed: Functional requirements, flexibility scenarios
- Produced: Proposed architectural elements (devtime or runtime) covering the functional requirements, proposed architectural *template* elements with assigned functionality sets

Guidelines and heuristics for flexibility

- Make a top-down decomposition of the system (hierarchically)
- Identify architectural elements, their interfaces, and relationships
- Cover in the decomposition the aspects: functions, data, processes, and UI
- For flexibility scenarios covering the change of a certain functionality, decompose the system in a way that the change is as local as possible.
- Build abstractions for recurring architectural elements, in particular when the level of granularity is not predetermined. Such abstractions are marked in the architecture model as *Template Elements* (see Section 4.3.1.3). Attach to a template element the functionality it abstracts from. The key idea is to defer the decision about the concrete decomposition until the point in time when the mechanisms to

achieve flexibility are clear. [Example: If many architectural elements for data access are found, a template element “data access” is defined and later-on, the concrete decomposition is made]

Activity: Existing Artifacts Analysis*Introduction*

Architecture design activities often take place in the context of system evolution. Thus, the architect has to consider the existing architectural decisions and implementation artifacts of the system, as they cannot be freely influenced. Changing a decision in architecture design might lead to high cost for the change of the implementation. Often, the architecture of existing artifacts is not explicitly known. Then, reverse engineering techniques are needed to recover architectural elements and decisions from the implementation.

Goals for flexibility

- Identify realized architectural decisions in existing artifacts, which hamper the achievement of flexibility
- Identify potential for creating flexibility without high effort and cost for changing existing implementation artifacts

Artifacts consumed and produced

- Consumed: Architecture models of existing system artifacts, flexibility scenarios, ideas for architectural realization of flexibility scenarios
- Produced: Judgment about feasibility of architectural ideas, alternative solutions

Guidelines and heuristics for flexibility

- Conduct a bottom-up analysis of the existing system artifacts and their architectural decisions: Make a mapping to requirements of the system under design
- Analyze flexibility mechanisms and business logic mapping in the existing system artifacts
- Analyze the impact of realizing drafted flexibility solutions based on the existing system artifacts (considering functions, data, processes, UI); identify architectural decisions that lead to high change impact
- Consider deviations from the planned concepts for flexibility, which are closer to the flexibility mechanisms realized in the existing system artifacts and thus allow easier realization

Activity: Realization of RunTime Quality Attributes

This activity cares about the achievement of quality attributes like performance or availability. It is quite similar to the realization of devtime quality attributes, but it deals with runtime architectural elements instead of devtime architectural elements. This activity is not in the direct scope of designing for flexibility; however it has a connection as design decisions made for runtime quality attributes might adversely impact flexibility. Thus, an iterative refinement between the realization of runtime and devtime quality attributes is necessary to guarantee that all relevant scenarios can be addressed.

Activity: Realization of DevTime Quality Attributes

This activity is the counterpart of the realization of runtime quality attributes. Flexibility is a devtime quality attribute and thus has to be addressed mainly in this activity. Due to this importance of the activity for our design process, we split this activity into three sub-activities which are described in the following.

Activity: RunTime - DevTime Mapping and Consolidation*Introduction*

Architecting means to define how a system can deliver the required functionality at runtime and how it can be developed at devtime. The respective architectural elements at both levels are not necessarily identical. As architecture design often starts at the runtime level (delivering the functionality is the foremost goal why software is developed) it has to be mapped at some point in time to the devtime level. This activity assumes that a runtime decomposition and potentially also the realization of runtime quality attributes has already been done. The key relationship between runtime and devtime architectural elements is that runtime elements are realized by devtime elements (not necessarily 1:1).

Goals for flexibility

- Map runtime architectural elements to devtime so that design for flexibility can be applied

Artifacts consumed and produced

- Consumed: Runtime architectural views
- Produced: Initial devtime architectural views

Guidelines and heuristics for flexibility

- Initially, map runtime components 1:1 to devtime modules
- Initially, map runtime data elements 1:1 to devtime modules
- Initially, map runtime process elements 1:1 to devtime modules
- Initially, map runtime UI elements 1:1 to devtime modules
- Identify multiple instantiations of elements at runtime and reduce at devtime to 1 single realizing element
- Identify common parts in resulting modules and factor out to separate modules (avoid redundancy at devtime level)

Activity: Application of Flexibility Mechanisms

Introduction

Flexibility is achieved when change requirements can be realized with minimal and local impact only. Flexibility mechanisms (see Section 4.3.1.2) are architectural mechanisms that support the localization of changes. Flexibility mechanisms typically introduce some kind of indirection, which allows the local change of particular system aspects. For reasons of complexity handling, we explicitly separate the *Application of Flexibility Mechanisms* step from the follow *Business Logic Mapping* step.

Goals for flexibility

- Identify and apply flexibility mechanisms in order to achieve the flexibility requirements

Artifacts consumed and produced

- Consumed: Initial functional decomposition at devtime level (produced by RT-DT Mapping or by Functional System Decomposition), flexibility scenarios
- Produced: Architectural decisions and views covering the selection and application of flexibility mechanisms, mainly at a business-logic-agnostic level (BLA)

Guidelines and heuristics for flexibility

- Explicitly address architectural elements that were tagged as *Template Elements*: Identify whether the application of flexibility mechanisms is necessary
- Analyze flexibility scenarios to see which aspects have to be separated from each other by means of flexibility mechanisms

- Cover aspects of functions, data, processes, and UI in flexibility considerations
- Select appropriate flexibility mechanisms (e.g. patterns) to be applied
- Identify the role of architectural elements in the flexibility mechanism (e.g. a workflow engine can represent a role and declaratively described workflows can represent another role)
- Identify whether the selected architectural mechanisms come with new infrastructure elements. Explicitly mark them as *Infrastructure Element* in the architecture
- Identify whether further abstractions of functional modules are meaningful: If yes, introduce further *Template Elements* abstracting from these elements
- Finish with an architecture model that is mainly business-logic-agnostic with respect to flexibility

Activity: Business Logic Mapping

Introduction

Business Logic Mapping is the step that integrates the functional decomposition of a system and the selection of flexibility mechanisms. Only when appropriate flexibility mechanisms are in place and when the business logic is appropriately distributed to architectural elements, a system is flexible. We introduced in previous activities *Template Elements*, which will now be concretely instantiated in order to have an appropriate mapping.

Goals for flexibility

- Identify a mapping of business logic to architectural elements defined by flexibility mechanisms so that the flexibility requirements can be adequately fulfilled

Artifacts consumed and produced

- Consumed: Flexibility scenarios, functional decomposition of the system, architectural decisions about flexibility mechanisms
- Produced: Concrete mapping of business logic to architectural element types

Guidelines and heuristics for flexibility

- Revisit all flexibility mechanisms built in and the related *Template Elements*.

- All template elements are instantiated with concrete instances of architectural elements covering a certain amount of business logic (so-called *Business Elements* are created) (e.g. for an abstract service concrete instances representing the business logic are created). This is only necessary where flexibility requirements demand the mapping otherwise the business logic mapping can be left open to later development activities.
- Distribute business logic over the template elements in a way that the resulting flexibility potential matches the flexibility requirements
- Identify appropriate granularity of architectural elements and of interfaces
- Business logic mapping can result in splitting or merging modules, creation of new modules, reallocation of responsibilities across modules
- Consider functions, data, processes, and UI as concrete forms of business logic, which have to be appropriately addressed, also in their interplay
- Finish with an architecture model that is mainly business-logic-specific with respect to flexibility

Activity: Selection and Application of Technologies

Introduction

Technologies play an important role for the achievement of flexibility as they often realize architectural mechanisms. Realizing an architectural mechanism often means that the *Infrastructure Elements* introduced by flexibility mechanisms are implemented by a technology, and potentially a technical frame for the realization of *Business Elements* is given.

Goals for flexibility

- Identify appropriate technologies supporting flexibility
- Apply technology in a way that optimally supports flexibility

Artifacts consumed and produced

- Consumed: Flexibility scenarios, architectural decisions about flexibility mechanisms
- Produced: Architectural decisions about technologies, (alternative) proposals about flexibility mechanisms

Guidelines and heuristics for flexibility

- For *Infrastructure Elements* identified during *Application of Flexibility Mechanisms*, typically a realization is needed. Often, existing technologies offer such realizations (e.g. commercial workflow engines realize architectural mechanism)
- Select technology with respect to their appropriateness to realize *Infrastructure Elements*. Alternatively, these elements have to be individually realized
- Make decisions about exact usage of technologies in the context of realizing flexibility
- Consider aspects of functions, data, processes, and UI
- Due to the availability of technological options, alternative flexibility mechanisms can be proposed, which can adequately replace selected ones

The design process activities as described are integrated via the process described in Figure 41 and the principles described in Section 5.3.2. This integration leads to several very intensive alignments of activities, which require iterative and incremental working towards architectural solutions. These particularly intensive alignments are highlighted in Figure 42. The architect should put specific focus on aligning the activities.

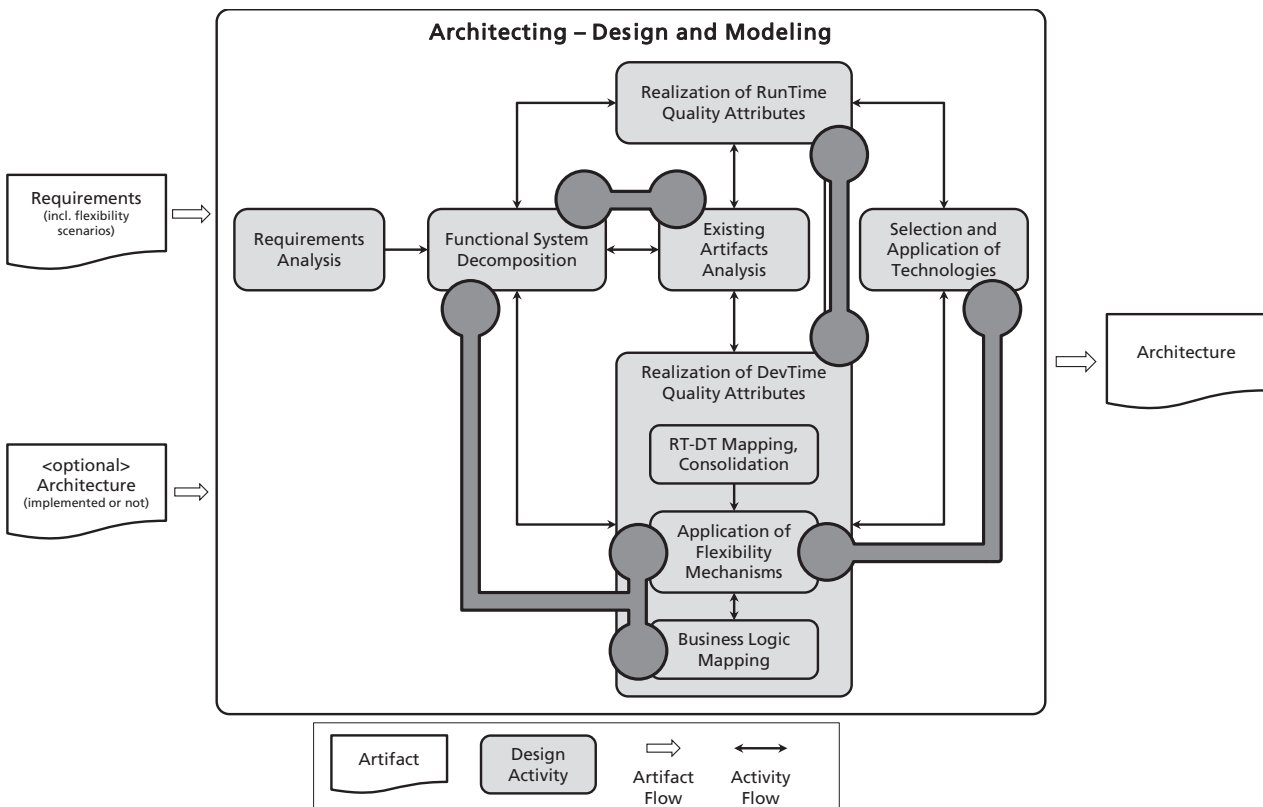


Figure 42: Architecting design process overview – key integrations

5.4 Measuring Flexibility with Tool Support

Figure 40 presents the overview on the methodical contributions to flexibility at architectural level. In the previous section, the constructive contributions for the *Design* activity are represented in form of activities, principles, guidelines, and heuristics. This section focuses on the analytical contributions, which are applied in the *Evaluate* activity.

While the evaluation of flexibility during architecture design is typically a manual and time-consuming task, we aim at automated analysis of flexibility. This facilitates the analysis of flexibility in much shorter cycles, it makes direct feedback to the architect possible, and it allows the architect concentrating on the design tasks with intermediate revisions on inadequate design decisions.

First, we explain the key ideas behind automated measurement and how it integrates into the engineering process in Section 5.4.1. Then, we outline the features of our tool developed for automated flexibility measurement and show an exemplary application in Section 5.4.2. Finally, we briefly describe the technical realization of the tool in Section 5.4.3.

5.4.1 Continuously Measuring Flexibility in Architecting

The key reason why to introduce automated measuring of flexibility is to drastically increase the frequency of measurement without distracting the architect from the design activities. While architecting as a whole aims at avoiding rework-intensive corrections at the implementation level, continuous measurement of quality attributes can help to avoid rework-intensive corrections of architectural design decisions at architecture level. The effectiveness of such near-real-time feedback has been shown also in other areas, for example for the avoidance of architecture compliance violations [Kno11].

Flexibility
measure

In Section 4.3.3.2, we formalized the ideas for measuring flexibility and introduced concepts for tool-supported measurement. The flexibility metric results are defined on a $[0, 1]$ scale, measuring the flexibility of an architecture with respect to a particular flexibility scenario. 1 means highest flexibility. In order to allow automated calculation of the flexibility metric results, we extended the architecture meta-model with additional information, captured in the *Change Impact View* (see Sections 4.3.3.3 and 4.4). As fully automated reasoning of change impacts for informal flexibility scenarios is not possible, we let architects model their considerations about the fulfillment of flexibility scenarios under design as part of the architecture model. This is expressed as change impacts on the architecture model and allows afterwards automated calculation of flexibility results.

Evaluation activity

Figure 43 depicts the *Evaluate* activity in the context of the overall design process. This step is fully automated and based on the architecture model created before, including the change impact view. The only task left to the architect in terms of the evaluation is the interpretation of the results and how to react to them in further design activities. Enhancing the architecture model with the change impact view becomes completely interwoven with the *Design* and *Model* activities.

Modeling change impact

Modeling the change impact views is relatively little additional effort. First, it captures only the information an architect has to reason about anyway when designing for flexibility since it covers the relevant aspects according to the definition of flexibility (and this is assumed to guide architects even more explicitly towards adequate flexibility solutions, as described in Chapter 7). Second, it can be done at the same point in time when the architect designs the flexibility solution. Third, being fully integrated in the architecture model in a tool, it can reuse existing model elements. Later on in the design and evaluation, the persisted change impacts can be revisited and reevaluated at any time.

In order to achieve near-real-time feedback, a close integration of the *Evaluate* activity with the *Design* and *Model* activities is necessary, as described above. Figure 41 shows the detailed activities of designing an architecture with focus on flexibility. Giving near-real-time feedback means that even working inside such an activity, the architect can get automated feedback on the current flexibility, as in the background the tool automatically calculates it for the current architecture model. The architect only has to make sure that the relevant change impact views are up-to-date. To achieve this, modeling architectural decisions and structures for flexibility should always be directly followed by modeling the change impact views.

In the following sections, we describe features, application, and realization of the tool for automated flexibility measurement.

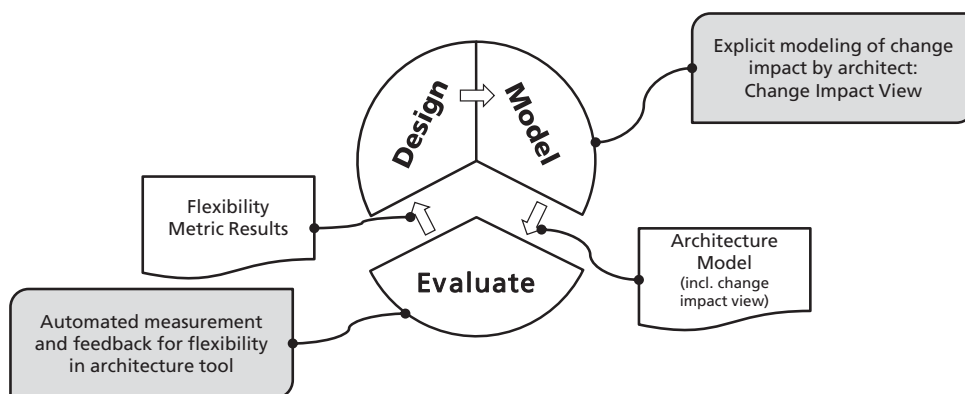


Figure 43: Key contributions to flexibility measurement

5.4.2 Features and Exemplary Application of the Tool

In order to demonstrate the feasibility of automated and continuous flexibility measuring and feedback, we developed a tool extension for the modeling tool Enterprise Architect (EA) [EA11a]. In Section 5.4.3, we describe the technical realization of the tool extension.

In this section, we focus on the functionality of our tool and demonstrate its application in the context of the example introduced in Section 1.3. Therefore, we first give a brief overview on the features of the tool. Then, we will revisit the example and give some further background information and assumptions. Finally, we illustrate with screenshots the application of the flexibility tool in the context of the example.

Overview of tool features

F1: Modeling of change impact views: Change impact views are introduced as a new type of architectural views in EA. In order to allow architects to easily model change impact views, the respective elements are provided as a toolbox, like for built-in notations and view types. In particular, flexibility scenarios and the impacts-relationship are introduced and allow graphical modeling of change impact.

F2: Representing flexibility-relevant data in the model: Change impact views and the respective architectural elements and relationships can be enhanced with more detailed information about the elements and the relationships, which are introduced in our flexibility metric. This is the prerequisite for more accurate flexibility prediction values. In particular, the type of the impacts-relationship (add / modify / delete) and the size of the impact (low / medium / high) can be modeled. The architectural elements impacted by potential changes can be described with their estimated or measured size (in LOC). This additional data is represented as tagged values in EA.

F3: Calculating flexibility automatically according to metrics: Having modeled the change impact view with the flexibility-relevant data, the flexibility tool can automatically calculate the flexibility metric for all flexibility scenarios in the architecture model. Therefore, the model is searched for all scenarios and the flexibility metrics are applied according to all metric configurations. The resulting flexibility values are between 1 (= best flexibility) and 0 (no flexibility).

F4: Representing flexibility metrics visually for user: The flexibility metrics results are visually represented to the user in two different ways. One is a small window which is always visible and represents the flexibility value for the flexibility scenario just selected in the modeling diagram. As soon as another scenario is clicked, the flexibility metric for this scenario is automatically calculated. Additionally, there is always the overall flexibility for all scenarios in the architecture model displayed. The other window is a main window in EA like a diagram. It presents an overview on all flexibility scenarios in the architecture model and also outlines the textual de-

scription. For each scenario, the flexibility metric value is displayed as well as the overall flexibility of the architecture with respect to all viewed flexibility scenarios. All flexibility values are colored in a traffic light style: Green depicting high flexibility, red depicting low flexibility.

F5: Configuring the flexibility metric: The calculation of the flexibility metric can be configured with a configuration window. Thereby, the change impact size for the predefined values can be adjusted (default: low = 10% change / medium = 30% change / high = 50% change). Further on, the flexibility metric can be adjusted in a way that the architect can define until which change impact size the flexibility is still considered to be 1 (default: less than 10 LOC impacted) and from which change impact size the flexibility is considered to be 0 (more than 10% of system size impacted). For details about the flexibility metric and the calculation see Chapter 4.

F6: Calibrating the flexibility model with scenario probabilities: As scenarios are not equally likely and the architect might like to try out different profiles of probabilities, flexibility scenarios can be tagged with expected probabilities. Then, the overall flexibility of the architecture is calculated by weighting the single flexibility scenarios' flexibility according to the scenario's probabilities.

Example revisited

In Section 1.3, we introduced a simplified architecture of a CheckIn system, as it could be found in the airline domain. We revisit this example to demonstrate the flexibility tool. The following extensions to the example have been made:

A **fourth flexibility scenario** was introduced:

FR4: Change the language in which the business process modeling is done to a more popular and powerful one

This flexibility scenario is quite difficult to handle as it requires changing the BP Engine and all business processes that are already realized (which is only 1 in our example).

The **change impact view** was **modeled**: We introduced the change impact diagram for the four scenarios and modeled the impacts. Additionally, the extra information about change impact was added (type and size of changes are annotated at the graphical impacts-relationship).

Element size was **added** for all architectural elements (estimated as LOC): We exemplarily estimated the figures as shown in Table 6.

<i>Element</i>	<i>Size [LOC]</i>
UI Engine	20.000
BP Engine	15.000
CheckIn (Descriptive Process)	20
Identify (Service)	3.000
Seating (Service)	3.000
Baggage (Service)	3.000

Table 6: Architecture example flexibility metrics - element sizes

In the following, we illustrate how the data is modeled with the flexibility tool and how the features can be accessed in the user interface.

Tool application for example

For this exemplary application, the flexibility tool is used in its default configuration (as can be also seen from Figure 46a). For easier understanding of the screenshots, we fade out the areas of EA, which in the respective screenshot are not of high relevance. For easier mapping to the features, we always provide a link to the feature ID, as introduced just above.

Figure 44a shows the structural architecture diagram of our example, as already introduced in Figure 5b. It can be seen that on selection of an architectural element, the element size can be entered as a tagged value [Feature F2].

Figure 44b shows how the change impact view is modeled. Using the change impact view toolbox, the relevant elements can be added: The flexibility scenarios can be added as elements and they can also be described in the notes field. Then, existing architectural elements can be dragged onto the diagram from the project browser. Finally, impacts-relationships can be drawn from flexibility scenarios to impacted architectural elements and the change impact type and change impact size can be set (see *Tagged Values* window) [Features F1 and F2].

Note that the diagram shown in Figure 44b only contains FR1, FR2, and FR3. FR4 is modeled in a different diagram. This allows easy scaling of change impact modeling and focusing on coherent sets of flexibility scenarios. As already described in Section 1.3, FR1 has very little impact on the business process description only. On the other hand, FR2 and FR3 have higher impact on several, also larger architectural elements. In particular, FR3 impacts the large infrastructure elements UI Engine and BP Engine.

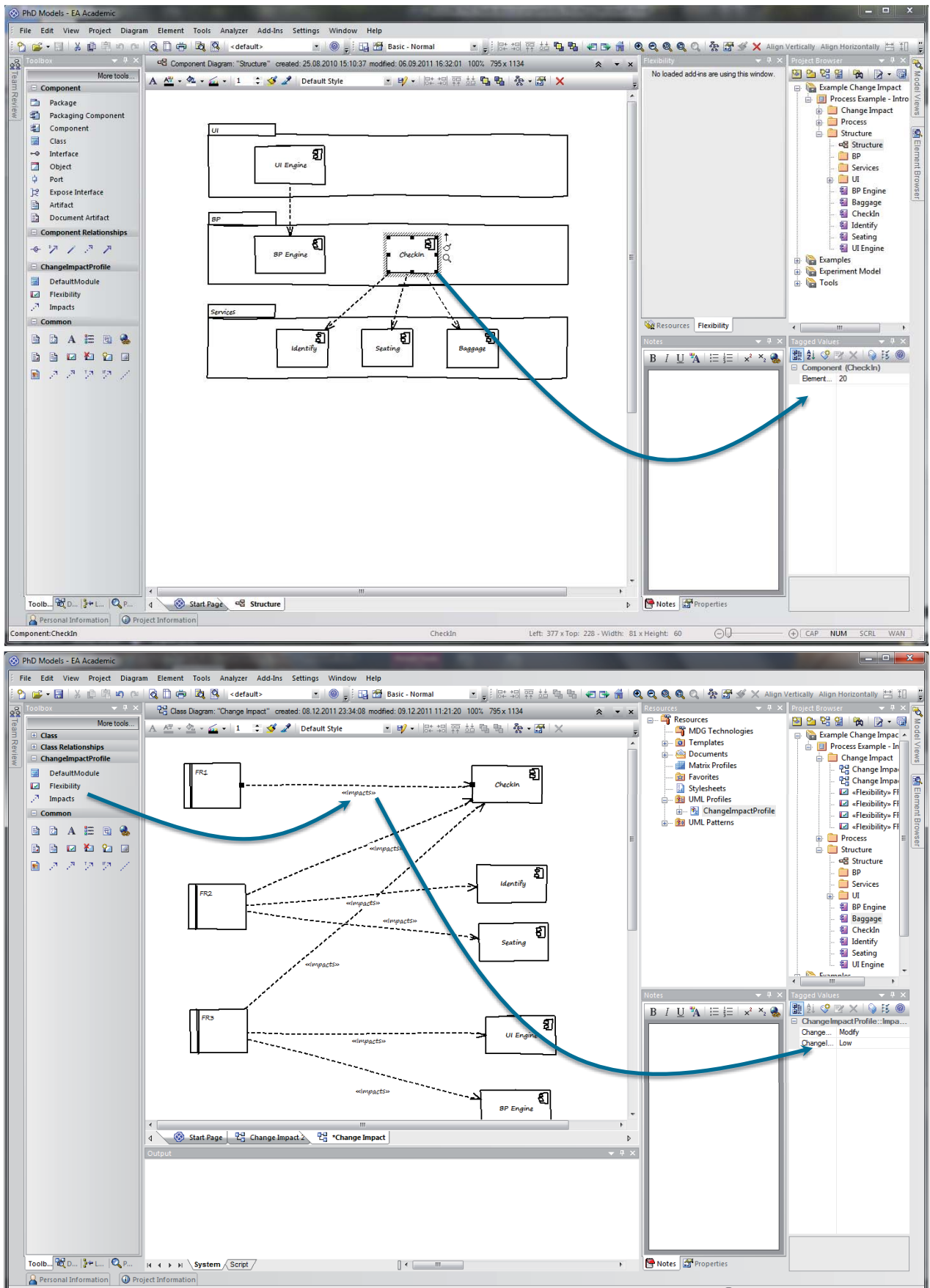


Figure 44: a) Modeling structural views in EA b) Modeling change impact in EA

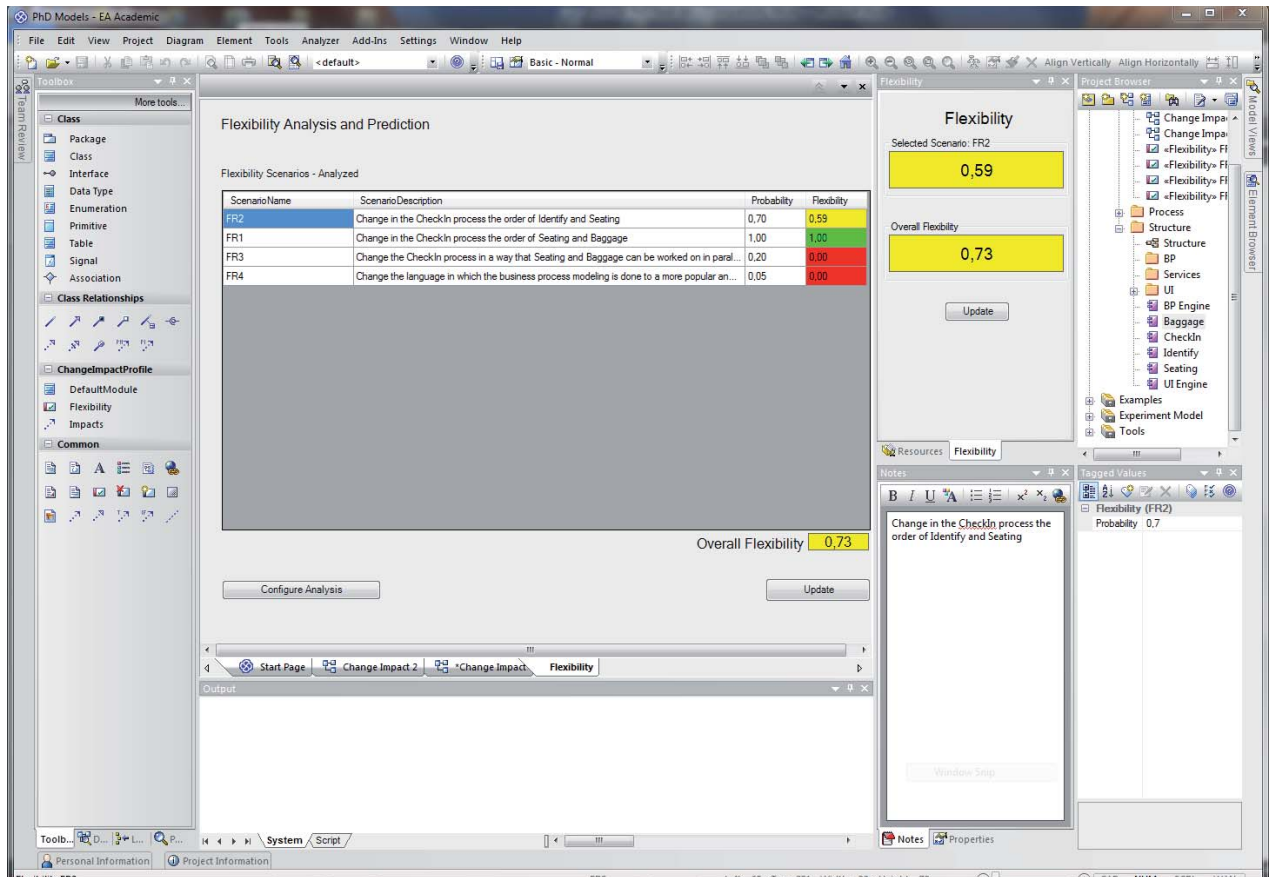


Figure 45: Flexibility evaluation results

Figure 45 depicts the results of the flexibility metrics calculation, as provided for the architect. The smaller docking window (visible also during modeling) on the right side presents the flexibility value for the currently selected flexibility scenario and the overall flexibility. The larger main window gives an overview on all flexibility scenarios and their respective flexibility values and depicts the assumed probabilities of flexibility scenarios [Features F3, F4, and F6].

In our example, also the flexibility result values show that our architecture is flexible with respect to FR1, and there is medium support for FR2, whereas our architecture is not flexible with respect to FR3 and FR4. This is expressed by both the calculated numbers and the traffic light colors. The probability values for the flexibility scenarios express that major changes like switching to a new programming language for business processes are rather unlikely compared to changing steps in a concrete business process. This is reflected in the overall flexibility of the architecture which can still be quite high although some unlikely scenarios are difficult to address. Due to the small probability values of FR3 and FR4, an architect might decide that the flexibility as achieved is good enough.

Figure 46a presents the configuration window for the flexibility tool and in particular the calculation of the flexibility metric values [Feature F5].

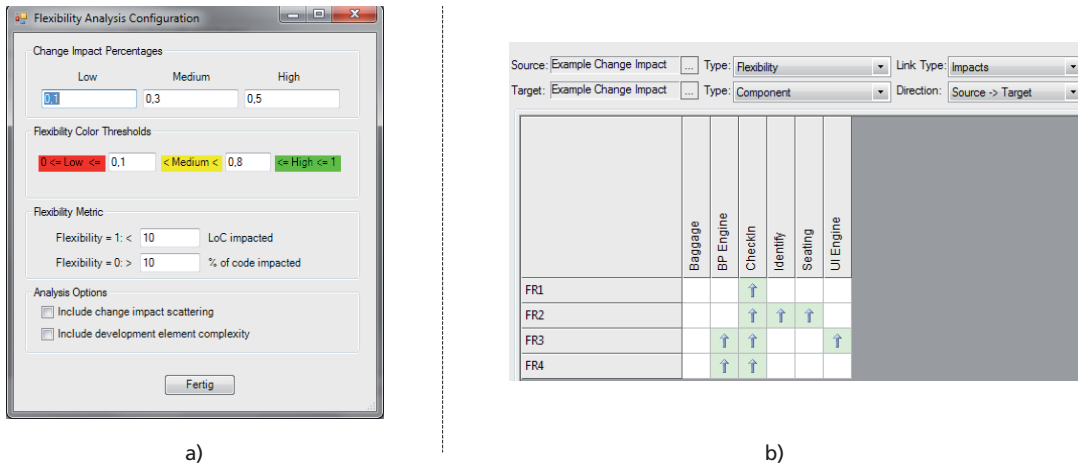


Figure 46: a) Flexibility tool configuration b) Matrix showing impacts-relationships

The values as shown are the default values. Figure 46b shows a matrix representation of the relationships between flexibility scenarios and the architectural elements impacted. It complements the change impact view with a more compact overview representation exposing less details.

5.4.3 Realization of the Flexibility-Tool

The flexibility tool is a prototypical implementation that was developed in the context of this thesis in order to show the automation potential of measuring flexibility in architecture design and to show the applicability of change impact modeling in an industry-accepted architecture modeling tool. In the previous section, the features and screenshots of the flexibility tool have been presented. In this section, the focus is on the technical realization of the tool.

EA AddIn Enterprise Architect (EA) [EA11a] is an UML modeling tool which is widely used in practice by architects and which is also the preferred tool at Fraunhofer IESE. It can be extended via an AddIn mechanism [EA11b] which was used to integrate the flexibility tool. On the one hand this allows contributing our tool to a well-established modeling platform, on the other hand it saves a lot of development effort due to the basic modeling facilities already provided. Further, the flexibility tool is well-integrated with other architecture tools of Fraunhofer IESE based on EA.

Development technology EA provides an extension API, which can be accessed via COM (Component Object Model). The AddIn is developed in C#, which allows easy publishing as another COM object. The AddIn is registered in the windows registry as a COM object, which allows EA to integrate it as an AddIn and provide access to it in the EA user interface.

Architectural overview Figure 47 depicts an overview of the architecture of the flexibility AddIn and how it relates to the core EA. In the following, the key architectural

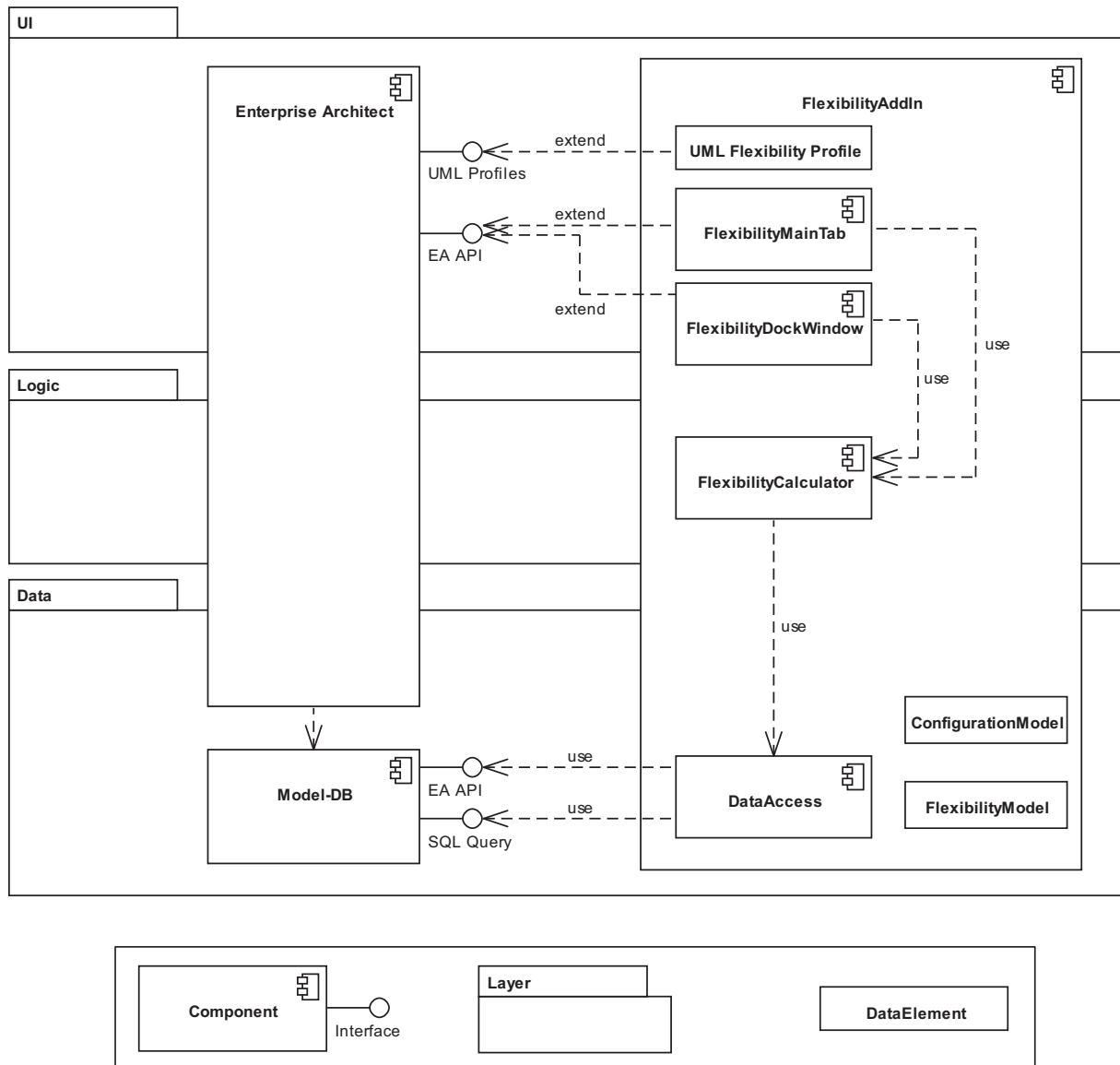


Figure 47: Architecture diagram for flexibility AddIn

decisions and components are outlined.

The architecture is organized along a three-layer architecture with UI, Logic, and Data. This layering holds true for both EA and the flexibility AddIn. EA is depicted as a black-box spanning all layers from UI to data. For the flexibility AddIn we provide the details on how it interacts with the interfaces of EA.

Modeling

The key foundation for automated measurement of flexibility in our tool is modeling an architecture, and in particular the change impact, according to our meta-model. We use the UML profile mechanism of EA and extend the available modeling language by our change impact view. We add in particular the flexibility scenario and the “impacts” relationship. The elements that are impacted are typically already there and we do not

have any restrictions on which development artifacts can be included as targets of change. This allows also modeling of inputs for model transformations as well as of descriptive artifacts like XML-based artifacts for inclusion in change impact analysis. Our extended profile appears as a new toolbox and allows easy modeling. Additionally, our meta-model and the flexibility metric require some more descriptions like the change probability, the type and impact size of change impacts, and the estimated size of the impacted elements (see Chapter 4). Such data is represented as tagged values in the model.

Flexibility
AddIn UI

The UI of the flexibility tool consists of mainly two UI elements: the main tab (*FlexibilityMainTab*) showing an overview of flexibility results for all scenarios and a docking window (*FlexibilityDockWindows*) that stays visible during modeling and exposes the flexibility of the currently selected scenario and the resulting overall flexibility. These UI elements are realized with Windows Forms technology. Further UI elements are already built-in into EA and only instantiated for our flexibility model (toolbox, tagged values, etc.).

Flexibility
metric
calculation

The component *FlexibilityCalculator* contains all the calculations of the flexibility metrics at all levels of aggregation. For the calculation, in particular the current configuration values which influence the metrics, are taken into account.

Data access
and data
elements

EA comes with a relational data model that can be accessed via two ways. First, there is an object-oriented interface, where data elements can be searched and the object tree can be navigated. Second, there is a SQL-based interface, which can directly query the underlying relational data model. For our AddIn, both ways of access are used. In case of large searches across multiple tables, SQL is used, which is typically a first step. Then, to get the data details of single data elements, the object interface is used. We define two own data types for the flexibility AddIn, which are used in all layers. One is collecting all the relevant data about flexibility calculations (*FlexibilityModel*), the other is collecting all the configuration data for the metrics calculation (*ConfigurationModel*).

5.5 Discussion

Summary

In this section, we contributed methodical support for the definition of flexible architectures. Our contributions enhance typical architecture definition methods with both, constructive guidance and analytical support for flexibility. These two methodical aspects are highly integrated as the analytical support aims at giving very quick feedback on the currently achieved level of flexibility, which an architect can directly use to revise his design decisions. The analytical support is realized as an AddIn for an architecture modeling tool, Enterprise Architect. By this, the contribu-

tions are integrated also from a tooling perspective, since the flexibility analysis works on the constructively created architecture model.

Our contributions do not replace existing architecture definition methods, but enhance existing ones with particularly detailed support for flexibility. Thus, our methodical contributions can be seen as a conceptual *Plugin* for architecture definition methods, focusing on flexibility. The key source for this methodical support is the conceptual foundation of flexibility defined in Chapter 4. There, the characteristics of flexibility are clearly defined, including a measurement model. These characteristics are used to define constructive guidance and analytical measurement support for architects.

Design for flexibility

Designing for flexibility we aim at constructing flexibility potential that matches the flexibility requirements. This requires defining an architecture that has adequate flexibility mechanisms in place and defines an adequate mapping of business logic to architectural elements. Only then, arriving changes can be conducted with minimal change impact. In order to allow defining such an architecture, we explicitly describe architectural design activities that care about design decisions supporting flexibility. Splitting these activities reduces the complexity the architect has to cope with and gives guidance about necessary steps and decisions. For more concrete guidance, we describe for each design activity the concrete artifacts on which the activity works and give heuristics on how to process.

Although these enhancements make the design process way more concrete for flexibility than in typical architecture design processes, it is still no straightforward process which could simply be automated. This is also visible by the fact that even no generally valid order of processing the design activities can be given. Architecture design stays a creative and challenging task for software architects, the methodical guidance can to some extent replace missing experience (as it makes best practices explicit). Additionally, adherence to this methodical support can lead to more uniformity of architecture design in a software development organization.

Measuring flexibility

Measuring flexibility is important during architecture design in order to check whether the flexibility potential achieved is adequate for the flexibility requirements. Automated measurement of flexibility is helpful for architects and creates minimal distraction from the design work. However, typical descriptions of flexibility requirements and architecture models do not allow the fully automated measurement of flexibility. Thus, we introduced the change impact views as an enhancement of the architecture meta-model. It is described by the architect and offers the data that is necessary to automatically calculate flexibility. A very important effect of the change impact view is that architects have to reason about flexibility in the necessary depth, which is expected to improve the flexibility potential.

The automated measurement is realized as an AddIn into Enterprise Architect, which allows fully integrated modeling of the architectural solution for flexibility and the change impact view. This emphasizes again our idea of extending architecture by a conceptual plugin for flexibility: Here it is even technically realized in that way. Tool-based support is in particular necessary and helpful for large-scale architecture models. Then, the model can hardly be captured mentally in all details and the tool-support allows working on separated areas. When architecture models are used to predict other quality attributes, too, this can be done on the same architecture model and then the architect can even detect tradeoffs in his analyses.

The implementation of the flexibility measurement tool is a prototype showing the feasibility of the technical realization and the practical applicability in the design process. Although the metric calculation itself is fully automated, there is a lot of further improvement and automation potential around the tool.

- The construction of the change impact views could be extended with automated proposals of change impacts, e.g. based on textual analyses of the flexibility scenarios.
- When the architecture evolves, it is currently necessary to manually adapt the change impact views. Automation support could identify potential impacts on change impact views and guide architects towards these changes.
- In the measurement tool, there could be a connection to the code base of current implementations in order to retrieve facts like the size of elements.
- An extension could take the cost for building in flexibility into account. That is, a metric would be defined which approximates cost of certain flexibility mechanisms and provides an integrated view with the achieved flexibility.

All these extension ideas provide further support for the architect and require further approximations and heuristics. They further reduce the manual workload for architects, but they do not lead to full automation of flexibility measurement.

The measurement approach mainly aims at improving the architecture design process. However, it can also be used as support in architecture evaluations in other contexts. Then, the modeling of the architecture and the change impact views might be additional effort, but it might pay off as it allows detailed analyses on the architecture model and can be used further in potential evolution activities.

Integrated approach in industrial context

Design for flexibility and measuring of flexibility are very closely integrated, both from a methodical and a tooling perspective. This offers architects full support for dealing with flexibility at architectural level. The approach as described is directly applicable in the industrial context (the tool is only a prototype and needs more robustness). The main prerequisites to introduce the approach into a software development organization is that this organization has a mature level of architecting capabilities. That is, architectural requirements have to be systematically captured and architectural decisions and views have to be documented and used. Then, the guidelines and the tool-support can be introduced and applied. One big advantage of this introduction is that all contributions do not require major changes of the previous processes. The tool-support does not impact the original architecture model but only adds the change impact views. The key advantage an organization can get is a better awareness of flexibility and a systematic approach to deal with flexibility which can be communicated to all stakeholders like customers, requirements engineers, architects, and developers. The cost to adopt the approach is relatively low as it does not require many changes. Additionally, it allows an incremental adoption, which can mean to first introduce the constructive part or even the analytical part. Chapter 7 summarizes the hypotheses about benefits of the approach and first evidences.

6 Flexibility in SOA-Based Information Systems

*"Being nonphysical, software parts can be far more flexible than physical parts. Therein lies the power of the medium beyond all others."
Paul Basset*

SOA-based information systems are one type of systems which are often not as flexible as needed and expected (see Chapter 1). We formulated as industry level goals I.G1 and I.G2 (see Section 1.3) of this thesis the goals to support architects in building flexible SOA-based systems, using in particular the flexibility potential of architecture mechanisms and technologies in SOA.

In Chapters 4 and 5, we introduced SOA-independent foundations and engineering support for achieving flexibility. In this chapter, we add SOA-specific aspects to the previous contributions. By narrowing down the scope of systems to the ones built according to SOA, more commonalities among challenges, solutions, and technologies can be found (see Figure 15). This allows providing guidance that is more specific to architects by describing typical challenges, solutions, and technologies that are used as material in an engineering process.

Conceptual plugin for flexibility in SOA

Thus, the contribution about flexibility-specifics in SOA as described in this chapter is a conceptual plugin for our flexibility engineering approach. The plugin comes with explicit knowledge of flexibility in SOA-based systems. Figure 48 depicts the aspects of flexibility related to SOA and shows how they extend our earlier contributions (in addition, the sections where to find the contributions are annotated). Figure 48 combines the representations of Figure 11 and Figure 15 in order to illustrate the relationships of contributions in detail.

The SOA-specific flexibility contributions depicted in Figure 48 also form the structure of this chapter. First, we detail the challenges around flexibility in Section 6.1. Then, we discuss architectural solutions for flexibility in Section 6.2 and technologies that realize these solutions in Section 6.3.

In this chapter, we consider mainly two perspectives on SOA: first, from the perspective of organizations acting as suppliers of software systems,

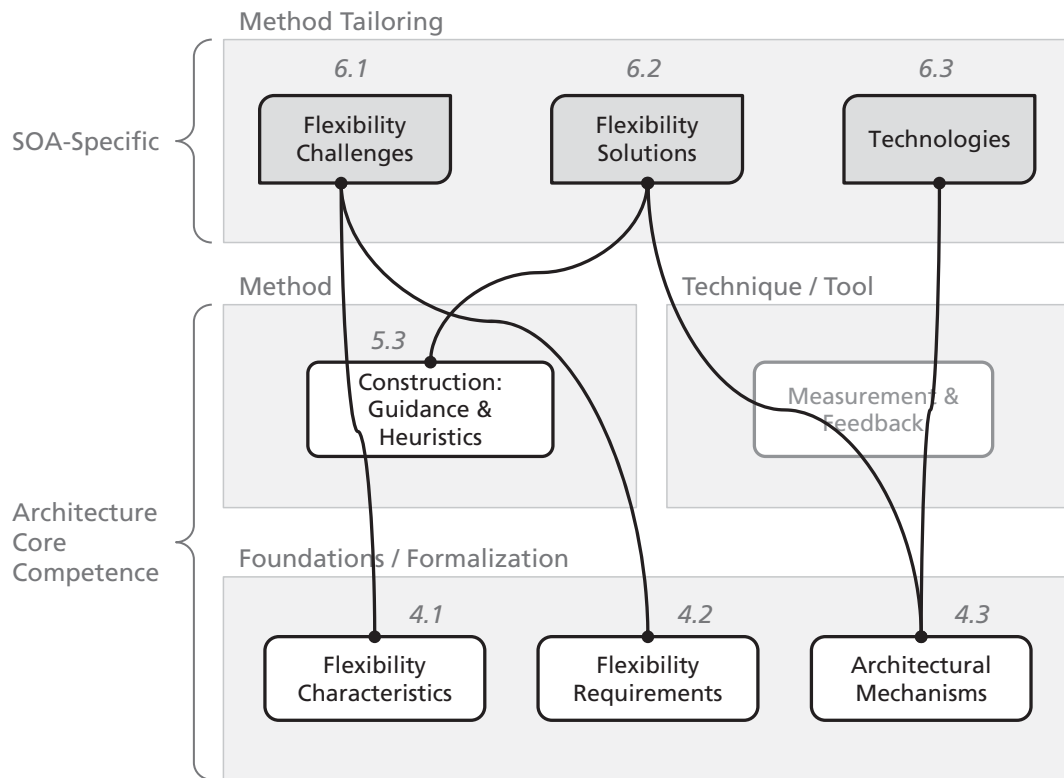


Figure 48: SOA-specific contributions around flexibility

which follow the SOA paradigm and are integrated in an application landscape; second, from the perspective of organizations integrating IT landscapes for customer organizations following the SOA paradigm. We explicitly do not consider market-place perspectives on SOA, which include more or less dynamic selection of services.

6.1 Challenges around Flexibility in SOA

Building SOA-based systems often has the goal to build highly flexible systems. However, also for SOA-based systems it holds true that universal flexibility is not possible. Thus, detailed knowledge of what needs to be flexible is necessary. In Section 6.1.1, we collect typical flexibility requirements as a guidance for eliciting flexibility scenarios. This collection makes the guidelines of Section 4.2 more concrete by incorporating the knowledge about SOA-based systems in practice. Of course, these are only typical flexibility requirements that are not intended to be complete.

Additionally, the typical settings where SOA is used come with characteristics that make it even more challenging to achieve flexibility. Such characteristics are described in Section 6.1.2.

6.1.1 Typical Flexibility Requirements

In Section 4.2.2, we describe questions to be applied for the elicitation of flexibility scenarios. With the knowledge of typical changes in the domain of SOA systems we can give support for answering the question “What has to change?”. It is important to note that the examples we contribute in the following (collected as experience from projects with industrial partners) are all at the business-logic-agnostic level (see Section 4.3.1.3), that is they do not refer to concrete business logic of a concrete system. For the elicitation of concrete flexibility scenarios, it is very valuable to cover business-logic-specific aspects for achieving flexibility.

We organize our typical flexibility requirements along the categories presented in Section 4.2.2 for the question “What has to change?”: *Functionality, Technology, External Systems*. *Quality* we leave out as no typical flexibility requirements have been experienced.

Functionality: Functions, data, processes, UI

- Changing the computational logic of a single service
- Changing business processes (order of activities, adding or deleting activities, consuming or producing data in different activities, ...)
- Introducing new business processes to be supported
- Extension or change of data structures for delivering new or changed data
- Changing the responsibility of services for certain data entities
- Changing the granularity of user interaction with the system (smaller or larger activities, more or less data provided or consumed in activity)
- Usage of services in processes with different interaction schemas or different needs in data

Technology: Integration or replacement of technologies

- Integration of service providers or service consumers using different implementation languages
- Integration of service providers or service consumers using different communication protocols
- Integration of service providers using different data management technologies
- Construction of applications using new portal and UI technologies

External systems: Integration with new or other external systems, changes due to changes in external system

- Extend the range of service consumers for new service consumers and delivery channels
- Integrate with a new service provider that uses a different data model
- Replace the service provider and use a similar service of a different provider
- Exchange a complete backend with one that has similar functionality (but maybe different interaction schemas, different data models, etc.)
- Follow the changes in the data model of service providers
- Provide functionality of an existing system as services to be used in a larger landscape or by other business processes

6.1.2 Characteristics Challenging Flexibility

SOA is a paradigm that aims at the realization and integration of large application landscapes in enterprise organizations, in particular aligning business and IT. From this particular context, characteristics arise that make achieving flexibility more difficult and that are helpful to know for architects during system design. We distinguish technical and organizational characteristics, experienced in projects with industrial customers.

Technical characteristics

- SOA is typically used in application landscapes with high inherent complexity
- The applications and building blocks being integrated in SOA systems are often heterogeneous with respect to implementation technologies, data management, architectural assumptions, etc.
- The applications and building blocks being integrated in SOA systems are often legacy systems which are hard to change
- The applications and building blocks being integrated in SOA systems often have a high complexity of data, in particular the underlying data models, assumptions about the usage of data, etc.
- The applications and building blocks being integrated in SOA systems are often not under the development control of the integrating company and thus have to be treated as black boxes
- The applications using services are often highly interactive systems which need strong tailoring to users' needs

Organizational characteristics

- In large application landscapes, the overall system is typically not under the control of a single development organization. Rather, different organizations are involved, which leads to limited impact on flexibility at the overall level
- Software services often have multiple users which might be even distributed over multiple organizational units or even organizations. This situation hampers the change of software systems which might be flexible from a technical perspective. In particular service interfaces need stability and cannot be changed easily

For these characteristics, no generally applicable solutions exist. However, the architect has to be aware of and recognize these characteristics in concrete projects in order to come up with applicable and adequate flexibility solutions.

6.2 Architectural Solutions for Flexibility in SOA

Although there is no universally agreed definition of SOA, a number of architectural principles and mechanisms (see Section 4.3.1) has emerged which are widely accepted. In this section, we describe the most important principles and mechanisms in SOA from the perspective of how they as architectural solutions contribute to flexibility (see Figure 15). We aim at answering the questions: “What does a SOA-based architecture look like?” and “Which flexibility potential comes with SOA?”.

While the general definition of SOA (see Definition 8) considers business and IT, we now focus on software architecture only. The summary of architectural principles and mechanisms defines an architectural style which we call the SOA style (see [GS94, BCK03, Lub07], see Section 2.2.4). Our key contribution here is to make the so-far implicit relationship of the SOA style and flexibility explicit by concretely describing which architectural principles are realized and which flexibility potential SOA bears. In that sense, it is a guideline for architects to make better use of the flexibility potential of SOA (see goals in Chapter 1).

Definition 17 SOA Style

The SOA style is an architectural style which describes architectural elements, their relationships and composition in SOA-based software systems. This description is formulated as a set of architectural principles and mechanisms.

In the following sections, we first outline the architectural principles behind SOA (6.2.1). Then, we describe in detail the architectural mecha-

nisms (6.2.2). Finally, we summarize key architectural considerations an architect has to make to use the flexibility potential in SOA (6.2.3).

6.2.1 Architectural Principles in SOA Supporting Flexibility

The following architectural principles of SOA contribute to the flexibility potential of SOA. They guide the architectural mechanisms described in the next section. Thus, they are only briefly introduced here; the flexibility potential is explained in the next section. These principles sketch an ideal solution, which in practice can often only be approximated.

- *Service Properties*: Services are self-contained, context-free, idempotent, technology-agnostic, coarse grained [HHV06, Jos07, KBS04] building blocks of software systems.
- *Orchestration and Composition*: Services can be composed to higher-level services [KBS04, Jos07, Erl06].
- *Loose Coupling*: Services and their consumers are only loosely coupled in terms of data aspects, technology aspects, timing aspects, etc. [Jos07].
- *Standardization*: Interoperability among services and their consumers at a technical level is supported by a standardization of description and communication protocols, which are often based on XML.
- *Descriptors*: For deployment, configuration, composition of services and service consumers, XML-based descriptors are used.

6.2.2 Architectural Mechanisms in SOA Supporting Flexibility

SOA can be described from an architectural perspective with several architectural mechanisms, which follow the architectural principles identified before. These mechanisms describe types of architectural elements and how they are related to each other (see Section 4.3.1). Additionally, architectural elements are stereotyped according to the schema of Section 4.3.1.3.

In this section, we contribute a characterization of architectural mechanisms of SOA with a focus on flexibility. We use a uniform description template which explains the key principles and decisions behind a mechanism, the architectural elements involved (see also Figure 17) and their relationships (illustrated with architectural views), and the contribution to the flexibility potential of a software system.

The *Template* elements are instantiated with concrete business logic mappings; the *Infrastructure* elements are either realized by means of available technologies (see Section 6.3) or else individually developed.

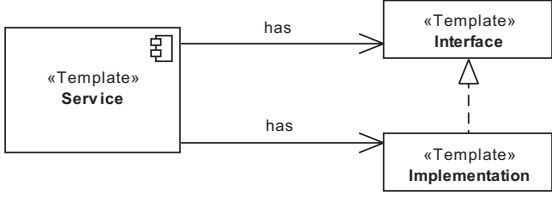
Mechanism	Service Concept
Key Principles & Decisions	<p>Services are designed to follow standard service properties (self-contained, context-free, technology-agnostic, coarse grained)</p> <p>Services make an interface public and hide their implementation</p> <p>Services can have multiple implementations, which provide different quality of service</p> <p>Interfaces can be differently defined: Interface vs. payload semantics; Interface semantics means that a service offers dedicated methods while payload semantics means that the service takes a document as input in which all actions and parameters are encoded</p>
Architectural Elements	 <pre> classDiagram class Service["«Template» Service"] class Interface["«Template» Interface"] class Implementation["«Template» Implementation"] Service --> Interface : has Service --> Implementation : has Implementation .. > Interface </pre>
Contribution to Flexibility Potential	<p>Internal realizations of services can be locally changed (algorithms, etc.)</p> <p>Internal data models are not exposed and can be changed locally</p> <p>New implementations of a service can be added without affecting consumers</p> <p>Internal technologies of services can be locally changed</p> <p>With payload semantics, the interface can be extended without affecting all service consumers</p> <p>Self-contained services encapsulate a certain amount of business-logic which can be changed locally in the service</p>

Table 7: SOA architectural mechanism: Service concept

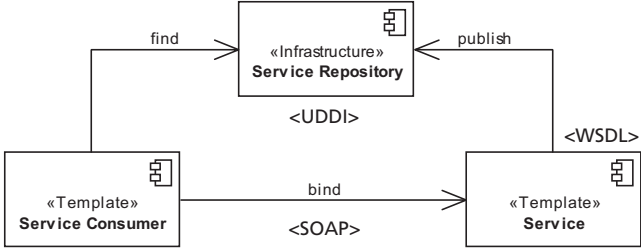
Mechanism	Basic Service Communication
Key Principles & Decisions	<p>Loose coupling: Service consumers and services are loosely coupled in the sense that a service consumer might not need to know concrete service providers implementing a service interface. Rather, concrete services can be identified via lookup in a service repository</p> <p>Loose coupling: Services can offer synchronous or asynchronous communication. With asynchronous service requests, the service consumer does not have to wait for the results. This only works when business logic is designed in a way that does not require immediate service results</p> <p>Standardization: For the communication among services, XML-based protocols exist in the Web Service technology environment. For example, WSDL is used to uniformly describe service interfaces and SOAP is used as a communication protocol for Web Services</p> <p>Services are available for service consumers in a distributed computing fashion</p>
Architectural Elements	 <pre> graph TD subgraph SC [«Template» Service Consumer] SC end subgraph SR [«Infrastructure» Service Repository] SR end subgraph S [«Template» Service] S end SC -- find --> SR SR -- publish --> S S -- «WSDL» --> SR SC -- bind --> S S -- «SOAP» --> SC </pre> <p>The diagram illustrates the Basic Service Communication architecture. It features three main components: a «Template» Service Consumer, an «Infrastructure» Service Repository, and a «Template» Service. The Service Consumer uses the 'find' operation to interact with the Service Repository. The Service Repository uses the 'publish' operation to interact with the Service. The Service provides «WSDL» (Web Service Description Language) to the Service Repository. The Service Consumer uses the 'bind' operation to interact with the Service, and the Service uses «SOAP» (Simple Object Access Protocol) to interact with the Service Consumer.</p>
Contribution to Flexibility Potential	<p>New implementations and even service providers of a service can be added without affecting consumers</p> <p>Internal technologies of services can be locally changed</p> <p>External systems can be integrated via exposing their functionality as services</p> <p>Integration of services using different implementation languages</p>

Table 8:

SOA architectural mechanism: Basic service communication

Mechanism	Service Typing
<p>Key Principles & Decisions</p>	<p>Orchestration and composition: Services can be orchestrated or composed, which means that a service realizes its functionality by consuming other services. This composition can be done hierarchically. Services that orchestrate other services are called molecular services, in contrast to atomic services [ANT+11].</p> <p>Introduction of different types of services: For separation of concerns, services can be typed in order to have clearer responsibilities for certain system aspects (data, functions) [HHV06]. Further aspects that might be encapsulated and separated are processes and UIs which we discuss in the next mechanism. Data aspects can be further separated, in data access and data transformation services.</p> <p>Service types can be organized in layers, which can be used to impose rules on access among different service types (e.g. function layer and data layer)</p>
<p>Architectural Elements</p>	<pre> graph TD subgraph Function_Layer [Function Layer] FS["«Template» Function Service"] FS --> FS end subgraph Data_Layer [Data Layer] DAS["«Template» Data Access Service"] DTS["«Template» Data Transformation Service"] end FS --> DAS FS --> DTS </pre>
<p>Contribution to Flexibility Potential</p>	<p>Orchestration of services allows hierarchically defining services at granularity levels that localize changes</p> <p>Separating function and data services allows changing of data persistency without impact on the processing functionality</p> <p>Data transformation services can help to localize changes of data structures in external systems</p> <p>Data transformation services can help to integrate new external systems with different data representations that have an impact on function services</p>

Table 9: SOA architectural mechanism: Service typing

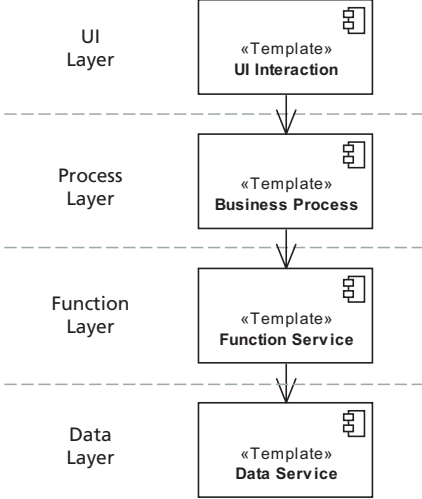
Mechanism	<i>Separation of Services, Process Logic, UIs</i>
Key Principles & Decisions	<p>Loose coupling: The functionality encapsulated in function services and data services is clearly decoupled from process logic using these services [HHV06]. Process logic, functionality and data are expected to have different change cycles.</p> <p>Loose coupling: UI interaction components are another layer of separation. Application frontends can be used to control business processes, but they can also directly access functionality or data via services.</p>
Architectural Elements	 <p>The diagram illustrates a four-layer architecture. On the left, the layers are labeled: UI Layer, Process Layer, Function Layer, and Data Layer. On the right, each layer contains a box representing a template: «Template» UI Interaction, «Template» Business Process, «Template» Function Service, and «Template» Data Service. Each box has a small icon in the top right corner. Vertical dashed lines separate the layers, and downward-pointing arrows connect the boxes from the UI Layer to the Process Layer, Process Layer to the Function Layer, and Function Layer to the Data Layer.</p>
Contribution to Flexibility Potential	<p>New business processes can be introduced with local change effort</p> <p>Business processes can be changed independently of function and data services (e.g. order of activities, adding or deleting activities)</p> <p>Changing the granularity of user interaction with the system (smaller or larger activities, more or less data provided or consumed in activity)</p> <p>Usage of services in processes with different interaction schemas</p>

Table 10:

SOA architectural mechanism: Separation of services, process logic, UIs

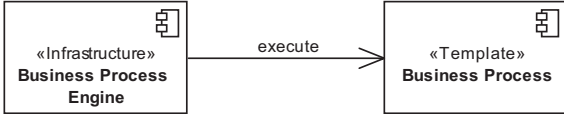
Mechanism	<i>Descriptive Process Logic</i>
Key Principles & Decisions	<p>Descriptors: Business processes are often not hard coded, but descriptively programmed in a business process or workflow language, e.g. in a language like BPEL or BPMN. A business process engine interprets the process description and executes it at runtime.</p> <p>Standardization: Business process languages are increasingly standardized, mostly with XML-based languages like BPEL.</p>
Architectural Elements	 <pre> graph LR A["«Infrastructure» Business Process Engine"] -- execute --> B["«Template» Business Process"] </pre> <p>The diagram illustrates the execution of a business process. On the left, a box labeled «Infrastructure» Business Process Engine contains a small icon of a computer monitor. An arrow labeled "execute" points from this box to a second box on the right labeled «Template» Business Process, which also contains the same computer monitor icon.</p>
Contribution to Flexibility Potential	<p>New business processes can be introduced with local change effort</p> <p>Business processes can be changed independently of function and data services (e.g. order of activities, adding or deleting activities)</p>

Table 11: SOA architectural mechanism: Descriptive process logic

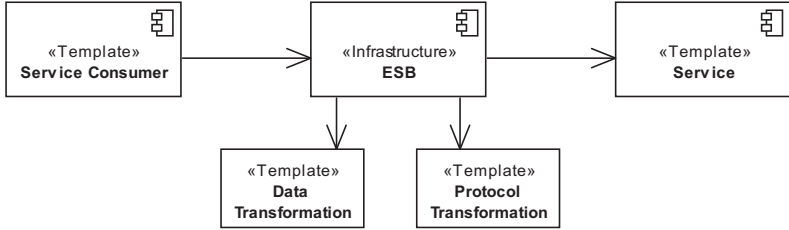
Mechanism	Enterprise Service Bus
Key Principles & Decisions	<p>Loose coupling: ESBs decouple service consumers from services in different aspects. First, ESBs can take over the responsibility of a service repository (see Table 8). Second, ESBs can realize full location transparency. Third, ESBs can execute complex data transformations and protocol transformations, so that service consumers do not have to care about harmonization.</p> <p>Orchestration and composition: ESBs can realize the communication about all types of service consumers and services. Service consumers can be other services, application frontends, or business process engines, etc.</p> <p>Descriptors: ESBs often work with descriptors for the representation of data transformations and protocol transformations.</p>
Architectural Elements	 <pre> graph LR SC[«Template» Service Consumer] --> ESB[«Infrastructure» ESB] ESB --> S[«Template» Service] ESB --> DT[«Template» Data Transformation] ESB --> PT[«Template» Protocol Transformation] </pre>
Contribution to Flexibility Potential	<p>New implementations and even service providers of a service can be added without affecting consumers</p> <p>Internal technologies of services can be locally changed</p> <p>External systems can be integrated via exposing their functionality as services</p> <p>Integration of services and service consumers using different implementation languages</p> <p>Integration of service providers or service consumers using different communication protocols</p> <p>Integration of service providers using different data management technologies</p> <p>Provide functionality of an existing system as services to be used in a larger landscape or by other business processes</p> <p>Integration with a new service provider that uses a different data model</p> <p>Extend the range of service consumers by new service consumers and delivery channels</p>

Table 12: SOA architectural mechanism: Enterprise Service Bus

The described architectural mechanisms in SOA are widely orthogonal to each other and can thus be combined for the design of a system architecture. We sketched for the mechanisms what they can contribute to the flexibility potential of a software system (landscape). This flexibility potential widely corresponds to the flexibility requirements sketched in Section 6.1.1. An architect can use this description of the flexibility potential in order to better align his architectural decisions with the flexibility requirements identified. However, as described in Sections 1.3 and 4.3.1.3, flexibility is only achieved if an appropriate business logic mapping is made. We discuss related aspects of importance in the next section.

6.2.3 Key Architectural Considerations for Flexibility in SOA

Despite the strong focus on *Services* in SOA, there are far more architectural decisions that have to be made in order to construct adequate systems with flexibility. As elaborated for architecture in general (see Section 4.3.1.3), the appropriate combination of architectural mechanisms and business logic mapping is the key to flexibility. Thus, an architect has to consider all the mechanisms described in the previous section and also how he can map the concrete business logic of the system under design.

Business logic mapping in SOA

In line with the architectural mechanisms, we sketched important aspects for business logic mapping and on which types of elements they are typically mapped. Functions, data, processes, and UI together form the business logic of a software system and have to be adequately decomposed to architectural elements. Data is an aspect of particular importance as it has to be considered when dealing with all the other aspects. UIs represent data and interact with users on data. Processes manage how data is retrieved, used, or stored. Functions work on data and process it. Another important aspect of data related to BLM is data consistency which is realized with technical concepts like transactions.

Flexibility requirements typically found (Section 6.1.1) are related to all these aspects of business logic. Thus, design for flexibility has to consider all of them, and in particular their interrelationships. Service design and in particular service granularity have to be brought in line with the other aspects.

Applying the contributions

The contributions of this thesis support an architect in making these architectural decisions with respect to flexibility. First, guidance is given for the elicitation of precise flexibility requirements (Sections 5.2, 6.1) with a specialization on SOA. Then, architects can apply the enhanced architecture design process (Section 5.3) with guidelines and heuristics aiming at flexibility. In particular the steps *Application of Flexibility Mechanisms*, *Business Logic Mapping*, and *Selection and Application of Technologies* (see Figure 41) are enhanced with the SOA-specific know-how, which is described in Sections 6.2 and 6.3. For the step *Business Logic Mapping*, available guidelines for service design [HHV06, Erl06, AGA+08] can be used for more detailed heuristics on service design.

6.3 Technologies Supporting Flexibility in SOA

SOA is a paradigm for the construction of software systems which offers many technologies to architects. These technologies typically realize architectural mechanisms and can be used as infrastructure components (see Figure 15). In this section, we sketch an overview of SOA technologies and map them to the architectural mechanisms explained in the

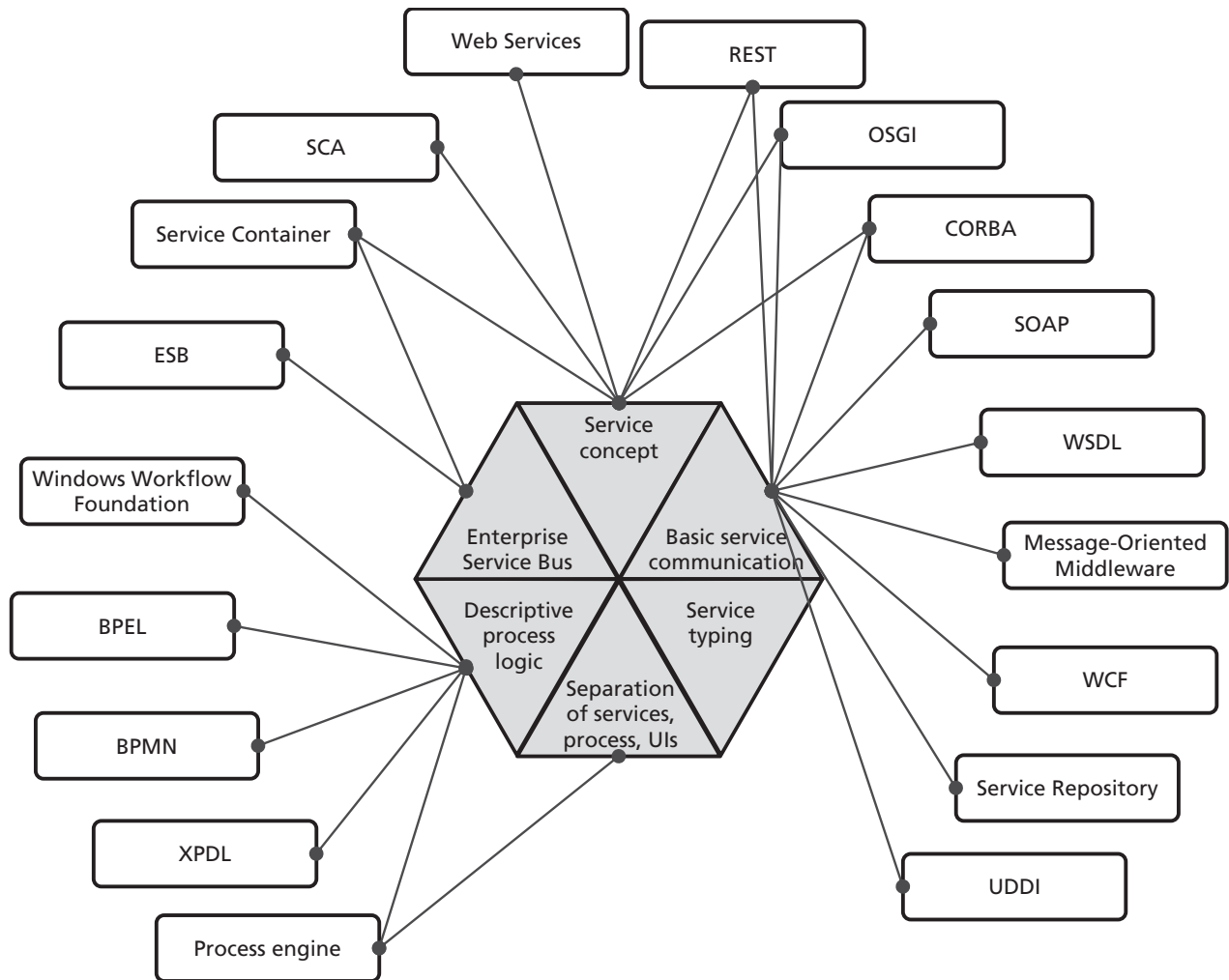


Figure 49: SOA architecture mechanisms mapped to SOA technologies

previous section. By that, our overview also supports architects in analyzing the flexibility potential of SOA technologies.

Figure 49 depicts the mapping of SOA architecture mechanisms to available SOA technologies and protocols. Typically, different technology alternatives are available to realize an architecture mechanism; often technologies realize only partial aspects of an architecture mechanism. Further, technologies often contribute to several architecture mechanisms. We do not consider individual products or brands in our overview; rather we depict classes of technologies or de-facto standards.

A noteworthy observation is that the architecture mechanisms *Service Typing* and *Separation of Services, Processes, UIs* are rarely covered by technologies. It is the responsibility of architects to come up with suitable architectural solutions in these areas, as they are mainly concerned with business logic mapping aspects.

7 Validation

*“Change alone is eternal, perpetual, immortal”
Arthur Schopenhauer*

This section describes the validation activities and results of this thesis. Therefore, we start with a description of validation objectives and the derived hypotheses in Section 7.1. Then, we describe in Section 7.2 a controlled experiment we conducted in order to show which effects the explicit modeling of change impact views during architecture design has on the resulting flexibility of the architecture. Besides this core contribution of our validation, we describe in Section 7.3 observations of applying parts of our method in industrial contexts.

7.1 Objectives and Hypotheses

We formulate the main goal of validation for our contributions in the GQM-goal representation [BD88].

*Analyze the flexibility method enhancements for the *purpose* of evaluation with a *focus* on effectiveness, efficiency, and applicability from the *perspective* of a software architect in the *context* of software architecture design.*

Our methodical contributions cover several aspects, derived from the research directions (R.D1-4) and research ideas (R.I1-4). Thus, also the validation has to cover the contributions in the respective areas. First, there is the conceptual model as the foundation for all methodical aspects, including the characterization and measurement of flexibility. Second, there is the constructive guidance towards flexibility in the architecture definition process. Third, there is the analytical part measuring flexibility and giving instant feedback to the architect, facilitated by tool support. Forth, there is the contribution with respect to SOA, making it easier to exploit the flexibility potential of SOA architectural mechanisms. For these areas of contributions, we derive hypotheses which are summarized in Table 13. These hypotheses are always in line with the goals and intentions of the contributions of this thesis. Our hypotheses cover the aspects **Validity**, **Effectiveness**, **Efficiency**, and **Applicability**.

R.I1: Conceptual Model	R.I2: Design	R.I3: Evaluate	R.I4: SOA
<p>H₁: Validity: The conceptual model is valid in describing the relationships between requirements, architecture, and implementation with respect to flexibility</p> <p>H₂: Validity: The provided measure for flexibility corresponds with intuition about flexibility</p> <p>H₃: Effectiveness: The conceptual model helps stakeholders in software development to better understand flexibility as a quality attribute</p>	<p>H₄: Effectiveness: Supported by the enhanced architecture method, architects define more flexible architectures</p> <p>H₅: Effectiveness: Supported by the guidelines for flexibility requirements, a better coverage of flexibility requirements can be achieved</p> <p>H₆: Efficiency: The additional cost for designing architectures according to the method is minimal compared to the cost of changes, which might be avoided</p> <p>H₇: Applicability: Practitioners can design architectural solutions for flexibility in the way proposed</p>	<p>H₈: Effectiveness: By explicitly describing how a flexibility solution for a particular scenario works, architects produce more flexible architectures</p> <p>H₉: Effectiveness: By getting continuous feedback on their architecture solutions, architects produce more flexible architectures</p> <p>H₁₀: Efficiency: The cost for explicitly describing architectural solutions for flexibility is minimal compared to the cost of changes, which might be avoided</p> <p>H₁₁: Applicability: Practitioners can describe architectural solutions for flexibility in the way proposed</p> <p>H₁₂: Effectiveness: Architects can judge flexibility of architectural solutions better when they explicitly model change impact</p>	<p>H₁₃: Effectiveness: For the paradigm SOA, a description of architectural mechanisms and their flexibility potential lead to better exploitation of the flexibility potential (more flexible architectures)</p>
<p>Project Experiences</p>		<p>Controlled Experiment</p>	
<p>H₁₄: Effectiveness: The introduced approach for flexibility extends existing architecture definition methods in a way that systems built with this method are more flexible and change requests can be, on average, conducted with less effort</p> <p>H₁₅: Effectiveness: The introduced approach for flexibility, together with the explicit descriptions of flexibility mechanisms in SOA, leads to SOA-based systems that are more flexible and change requests can be, on average, conducted with less effort</p>			
<p>Quantitative Results</p>		<p>Qualitative Results</p>	
<p>Source of Results</p>			

Table 13: Hypotheses for the areas of contributions

The hypotheses with respect to effectiveness and efficiency are comparatively formulated. We compare our contributions to typical architecture design methods, which do not include the contributions of this thesis. For reasons of clarity, we do not repeat in each hypothesis "... compared to ...".

In Table 13, a full spectrum of hypotheses is listed, from purely internal character (H₁, H₂) to purely external character (H₁₄, H₁₅). For the validation of these hypotheses, we conducted a controlled experiment and applied parts of the contributions in projects with industrial customers. We came up with both quantitative and qualitative results. For an in-depth validation contribution, we focus on one of the most fundamental hypotheses of our methodical contribution: the effectiveness of explicitly modeling change impacts during architecture design (H₈, highlighted in Table 13). We conducted a controlled experiment to evaluate this hypothesis and came up with quantitative data supporting H₈ with statistical significance (see Section 7.2). Additionally, we collected qualitative results indicating support for our hypotheses in projects with industrial customers (H₃, H₄, H₅, and H₆, see Section 7.3) and the experiment (H₁₁, see Section 7.2.4).

The scope of our hypotheses ends with H_{14} and H_{15} , where the effort of change requests, as expressed in the definition of flexibility, is considered. H_{14} and H_{15} contain basically two aspects: From an architect's perspective, the effectiveness in the sense that a flexible architecture is achieved. From a developer's perspective, the efficiency in the sense that incoming changes can be conducted with little effort. In our motivation and problem description (see Chapter 1), the scope was even broader: We discussed business opportunities resulting from the possibility to conduct software changes quickly, meaning to have adequate flexibility. To achieve this, many other prerequisites are necessary so that we do not include such effects in our hypotheses for the method.

7.2 Controlled Experiment

For the validation of Hypothesis H_8 *“By explicitly describing how a flexibility solution for a particular scenario works, architects produce more flexible architectures”*, we conducted a controlled experiment.

In the following, we describe context (Section 7.2.1), setup (Section 7.2.2), and analysis results (Section 7.2.3) of the experiment. Then, we discuss the results (Section 7.2.4) and threats to validity (Section 7.2.5).

7.2.1 Context of the Experiment

Practical course: TU KL, IESE	The experiment took place in a practical course for master students at the Technical University of Kaiserslautern (TU KL). The practical course was supervised in cooperation with Fraunhofer IESE in winter semester 2011/2012.
Capstone: John Deere	The practical course was a so-called Capstone Project with a real customer from industry, John Deere. That is, John Deere cooperates in the course and provides requirements for a smaller product and the students apply software engineering activities (requirements engineering, user-interface design, architecting, implementation, quality assurance, and project management) in order to realize the requested product. The students are assigned to specific roles like project manager, architect, or developer. In total, 17 master students participated in the course. Fraunhofer IESE researchers supported the students with tutorials on the methods to be applied and continuous feedback on the results produced.
Development support system	The system under development is a tool supporting the agile development approach at John Deere. It is a dashboard aggregating development information like the current status of agile development projects (user stories, burn-down charts, etc.) on a screen for distributed development teams.

Iterative development	This system was developed with an iterative development approach splitting the overall development into three iterations. Each iteration was supposed to produce a running system demonstrating a certain amount of functionality.
Experiment: Build on project architecture	All the participants of the practical course were included in the experiment and it bases on the system under development. The main reason for this is that the students are already familiar with the system domain and have a precise idea about the architecture of the system. Due to the incremental approach, an architecture document of sufficient quality was available from iteration 1. Thus, our experiment could start without detailed explanation about the system and so it could focus on the experimental tasks.

In the following, we explain in detail how the experiment was set up.

7.2.2 Setup of the Experiment

In order to explain the setup of the experiment, we will first start with the formulation of scientific hypotheses and with how they are operationalized. Then, more information on the participants, the experimental design, procedures, tasks, and materials is provided.

7.2.2.1 Scientific Hypotheses

The hypothesis to be tested in the experiment is: *“By explicitly describing how a flexibility solution for a particular scenario works, architects produce more flexible architectures”*. The comparison is against architecture design following typical design approaches without modeling change impact views.

The major idea behind the experiment is to compare two groups, A and B, designing architectures for flexibility, one with the technique explicitly describing flexibility solutions (group B, the treatment group) (see Section 5.4.1) and one without (group A, the control group). The detailed description of the experimental design can be found in Section 7.2.2.2. In the experiment, all participants acted in the role of an architect, independent of the role in the overall course.

As a basis for the comparison of solutions, we need a clear metric for flexibility which can be calculated for the experiment results produced by the participants. We use the flexibility metric defined in Chapter 4 which defines flexibility on a $[0, 1]$ range with an interval scale.

Derived from our hypothesis stated above, we define the scientific null hypothesis and the corresponding alternative hypothesis. We formulate

our hypotheses in a directed way. μ_A and μ_B denote the arithmetic mean of the flexibility achieved in group A and group B.

$$H_{8,0}: \mu_A \geq \mu_B$$

$$H_{8,1}: \mu_A < \mu_B$$

Our hypotheses are tested at a confidence level of $\alpha = 0.05$.

Besides expecting a statistically significant difference, we expect a difference of more than 0.1 on the flexibility scale [0, 1], which should be calculated by means of the effect size d (Cohen's d).

We set up our experiment with a concrete architecture definition task, in which the participants are asked to change the input architecture in a way that it offers best flexibility for three provided flexibility scenarios. The details on the tasks are described in Section 7.2.2.5.

In the following section, the experimental design will be described in detail.

7.2.2.2 Experimental Design

Groups A & B According to the idea described in the previous section, we designed an experiment with two groups. Group A conducts architectural design without explicitly modeling the change impact of flexibility solutions, group B parallelizes the activities of designing architectural solutions and the explicit modeling of their change impact. The idea of explicitly modeling change impact in the architecture model was not known to the participants before. Working with change impact views is expected to lead to strong learn effects. Thus, we decided not to follow a cross-design for the experiment.

Differences Figure 50 graphically depicts the experiment design. Input for both groups was an architecture (in version v_i) and a flexibility scenario. Then the task was to change the architecture in a way that it was flexible with respect to the flexibility scenario. As an additional input, the architectural modeling notation was explained. Group B also got the task to model change impact and got guidance by a description of the notation.

In order to be able to compare the results of both groups A and B, group A was asked, after finishing architectural design, to estimate the change impact and to also document it. For this, group A got as input the description of the change impact notation after finishing the architecture design. Thus, we had the same result artifacts from both groups, but they were created according to different procedures: an updated version of the architecture (v_{i+1}) and a diagram depicting the change impact (see

Figure 50). The results were created independently for three flexibility scenarios.

Group assignment The participants were randomly assigned to groups A (8 participants) and B (9 participants) and none of the participants was known to the experiment supervisor before. The participants did not know about the differences among the groups.

7.2.2.3 Participants

The participants of the experiment were 17 master students of computer science (10), software engineering (6), and telecommunications (1). The students were largely in their third semester of the master studies and aged between 23 to 30 ($\mu=25.3$; $\sigma=1.9$). They participated in a practical course of the Technical University of Kaiserslautern (TU KL) which was mainly supervised by Fraunhofer IESE. All participated on a voluntary basis and received no compensation.

14 out of 17 students had participated in architecture lectures at university before. In the practical course, they acted as requirements engineers (4), UI designers (3), architects (4), developers (2), testers (2), and project managers (2).

In a prebriefing questionnaire (see Appendix B), we asked the students

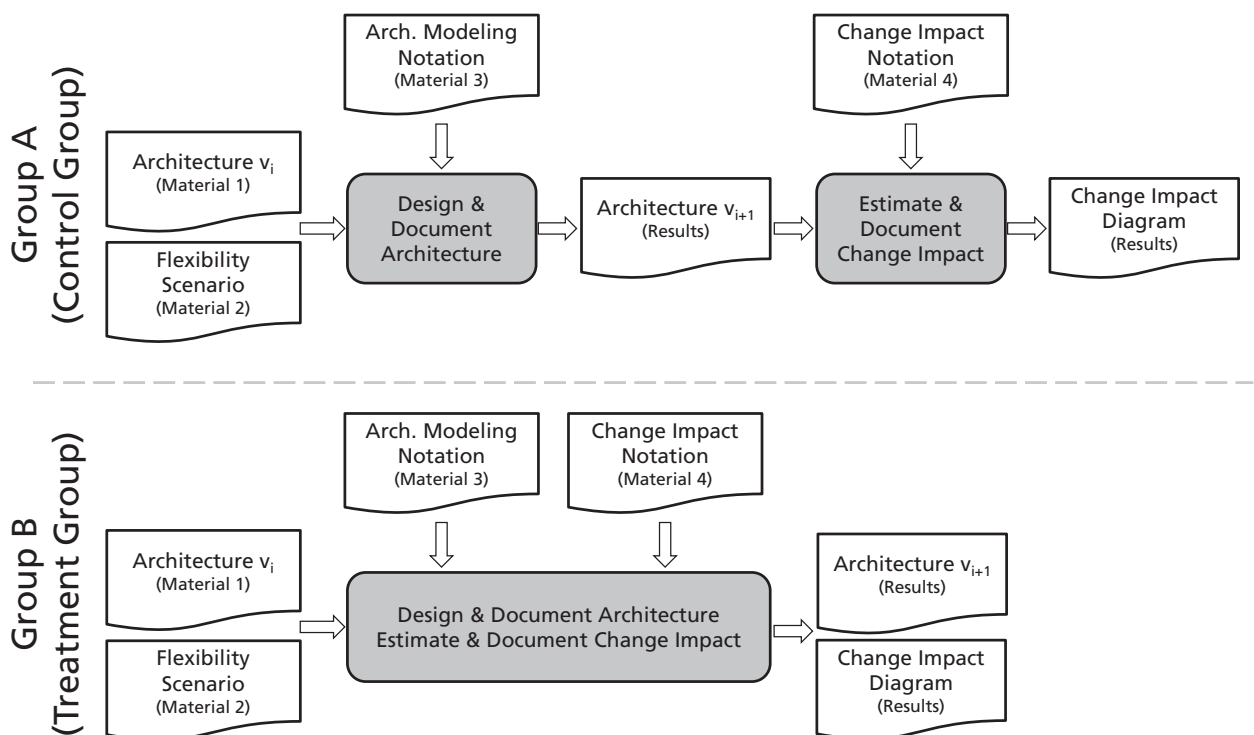


Figure 50: Experimental design

about their background and previous experience in relevant areas: Experience in development projects, reading and using UML, and architecting. The answers of participants in group A and B were quite similar. Thus, a similar influence of the background on the experiment results can be expected.

7.2.2.4 Experimental Procedures

Nearly all participants conducted the experiment in one afternoon, in a allocated time slot of 90 minutes. Three participants conducted the experiment on the next morning. Thus, the overall experiment was conducted with all participants in two subsequent days. All participants of one experiment run started at the same time. In the beginning, the setting was described by the supervisor; the detailed procedure was printed in the distributed material.

Preparation	The experiment started with a preparation phase which mainly consisted of reading and understanding the material. In particular, definitions on the background of flexibility and what it means to design for flexibility were given. Additionally, the role of the experiment in the practical course and the role of the participant in the experiment were clarified. Then, a briefing questionnaire was filled in by the participants, asking for information like their age or background knowledge.
Execution	In the execution phase, the participants worked on the tasks given (see Section 7.2.2.5) and produced an updated architecture and a diagram showing the change impact for three flexibility scenarios. The execution phase is also depicted in Figure 50. For group A, the task to model change impact and the respective notation guidelines were distributed after the architecture model had already been finished.
Finalization	In the finalization phase, the participants filled in a debriefing questionnaire, asking for example for the perceived difficulty of tasks and the perception of change impact views (see Appendix B).
Time constraints	The maximum execution time for the overall experiment was limited to 90 minutes. Participants of group A on average needed 71 minutes, participants of group B needed 65 minutes.

7.2.2.5 Experimental Task

Design for flexibility	In the experiment, the participants acted in the role of a software architect and had to conduct architecture design tasks. The main task they were asked to perform was to extend the existing architecture of the system (as resulting from the first iteration of the course) in a way so that it is flexible with respect to three given flexibility requirements. These flexibility requirements were specified as flexibility scenarios (as
------------------------	---

described in Chapter 4). For reasons of simplicity, the participants had to design three independent solutions for the three flexibility scenarios.

Architecture process The architecture design process was not prescribed to the participants. They got an introduction to architecture design in a tutorial at the beginning of the course; additionally most of them had attended lectures on software architecture before.

Architecture results For the architectural results to be produced, a simple notation with examples was given. The participants had to produce two artifact types (according to the experiment design shown in Figure 50). The first was a description of the resulting system following a structural component notation. The second was a description of change impact according to the notation introduced in this thesis. Although the flexibility metric and the change impact notation are originally defined for development time artifacts, we used the component notation that the students also used in their documents. This is a simplification of architecture modeling, which means a unification of runtime and development time elements. This was mainly done to allow the students to stick to their notations previously used. Even in industrial practice, this is a simplification that is often fully valid when runtime components are one-to-one implemented as development time elements.

The original task description is included in Appendix B. The material being processed in the task is described in the next section.

7.2.2.6 Materials

All necessary information was given in the form of experiment preparation and execution material. The architecture to be worked on was developed by the students in the course before and thus familiar to all the students. Besides this, they were also allowed to look into the architecture documentation of the system under development during the experiment if needed.

The following material was given to the participants in the experiment for conducting the tasks (see also Appendix B for the original material).

- *Architecture Documentation Input* (Material 1): The input architecture which had to be changed for making it more flexible. The input architecture was completely based on the architecture defined by the students in the course. It was reduced to one view, following the notation also described in Material 3.
- *Architecture Flexibility Requirements* (Material 2): Three flexibility scenarios describing the flexibility requirements, for which the architecture had to be made flexible. The flexibility requirements are not given by the customer, but they are invented by the experiment designer based on his knowledge of similar systems.

- *Architecture Modeling Notation & Example* (Material 3): The explanation of the simple modeling notation needed for the experiment. Material 3 focused on the component diagrams. The notation was intentionally kept simple and close to the notation the students had used.
- *Architecture Change Impact Notation & Example* (Material 4): The explanation of the simple modeling notation for change impact, as introduced in this thesis.
- *Original architecture document*: The participants were allowed to have a look into the architecture document of the system under development. This allowed us to keep the input documents brief and clear as the students were able to check all questions in the original document.

Besides this key task materials, there was the explanation of the experiment procedures (as described in Section 7.2.2.4), the questionnaires, and the task description.

After describing the setup of the experiment, the next section presents the analysis and results of the experiment.

7.2.3 Analysis and Results

We analyzed the experiment results in detail and describe in this section the procedure for data analysis, the basic data achieved, and statistical testing for our scientific hypotheses (see Section 7.2.2.1). Finally, we provide information on the results of the debriefing questionnaire.

7.2.3.1 Data Analysis Procedure

Variables The main independent variable in the experiment is the assignment to group A or group B, which means to design architecture either according to a standard design method or else with the additional usage of

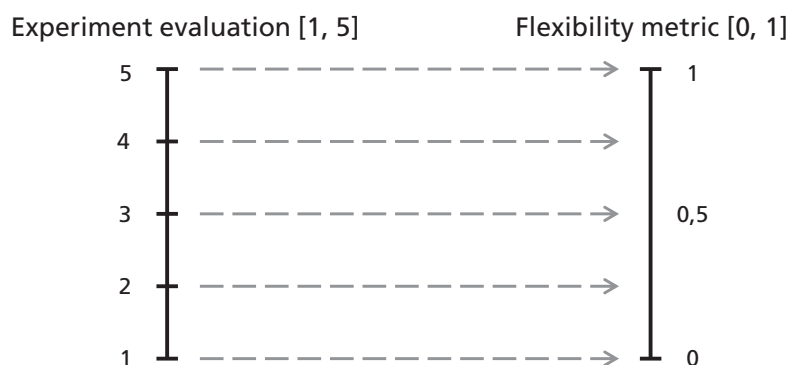


Figure 51: Measuring flexibility in the experimental results

change impact views. The key dependent variable in the experiment is the resulting flexibility of the participants' architectural solutions for the three scenarios given as input for design.

Measuring

Thus, the most important step in the data analysis is to evaluate the participants' results and to derive for each addressed scenario a flexibility result in the $[0, 1]$ scale as defined in Section 4. Initially, there was the plan to fully calculate the flexibility metric according to the participants output. Thus, the notation was given in a detailed way, covering also the relative size of touched elements and the impact on them. However, it turned out that the participants quite often did not fill in all data according to the notation. Thus, we had to slightly adapt the evaluation of the architecture results to a more expert-based procedure. In order to keep the expert estimation manageable, we introduced a five-point Likert scale $[1, 5]$, and mapped it to the $[0, 1]$ flexibility scale in the way depicted in Figure 51:

The Likert scale was introduced as it provides a manageable number of discrete choices among which an expert can decide. The $[1, 5]$ scale was selected in order to ensure consistency with all the other scales in the evaluation.

Filtering

As we expected that not all participants would provide adequate architectural results, we also introduced a filter mechanism. Each architectural result (per participant per scenario) was first rated for meaningfulness of the architectural solution. This is intentionally completely independent of the adequacy for flexibility scenarios. Rather it judges whether the architecture as such is understandable and shows that the participant understood the task to be solved. We rated the architectural solution in the component diagram and the change impact view independently of each other on a five-point Likert scale (1=completely inappropriate / 5=completely appropriate). Only if for both checks a value of at least 3 was achieved, this solution was taken into account for estimating the flexibility.

The method owner carried out the evaluation of flexibility by determining metric values. In order to minimize the risk of a single evaluation, we conducted two checks: First, we conducted the evaluation again after four weeks; second, we conducted an evaluation where we did not check the resulting flexibility, but the relative improvement of flexibility with respect to the input architecture. Both checks did not reveal significant divergences.

In the following sections we will first describe the basic results and then describe the statistical test for our scientific hypotheses.

7.2.3.2 Basic Results

For both the groups A and B, the ratio of valid (not filtered) solutions was quite similar, but in both cases below 50%. For each group we got 11 valid solutions. The number of valid solutions for each scenario is depicted in Table 14. We can observe a nearly equal distribution of valid results, which indicates that there was not a single overly complicated scenario.

	<i>Scenario 1</i>	<i>Scenario 2</i>	<i>Scenario 3</i>
Group A (n=11)	2	5	4
Group B (n=11)	4	4	3

Table 14: Number of valid results per scenario and group

The main reasons for invalid solutions were architecture descriptions (component views) which were not appropriate in any way for the flexibility scenario, or which were seriously incomplete. Although the participants did not always exactly follow the notation for the change impact view (missing data), the change impact views in general were much more appropriate. Nearly no result was filtered out due to inappropriate change impact views. This indicates that the participants got used to it quickly.

There is a correlation that participants who produced valid results did this consistently for all scenarios. This is not true for all cases but there is a strong tendency.

As the number of valid results per scenario group is rather small, we merged all results for group A and group B respectively. The individual results for flexibility values are listed for group A and group B in Table 15.

	<i>Flexibility Values</i>
Group A (n=11)	0.50 0.75 0.50 0.50 0.50 0.50 0.50 0.75 0.50 0.25 0.50
Group B (n=11)	0.75 1.00 1.00 1.00 0.75 0.75 0.50 0.50 0.75 1.00 0.75

Table 15: Flexibility values achieved (valid ones only) per group

7.2.3.3 Testing Hypothesis H₈

Mann-Whitney-U-Test

The statistical testing of hypothesis H₈ was done using the Mann-Whitney-U test (one-tailed). It is appropriate for small data sets like in our experiment and it does not require a normal distribution of the data

(we do not have a normal distribution). As introduced in Section 7.2.2.1, we formulated a directed null hypothesis. Our confidence level for rejecting the null hypothesis is 95% ($\alpha = 0.05$).

H_g: “By explicitly describing how a flexibility solution for a particular scenario works, architects produce more flexible architectures”

Test input: Individual flexibility result values for architecture design tasks, organized in group A and group B (see Table 15)

H_{g,1}: $\mu_A < \mu_B$

H_{g,0}: $\mu_A \geq \mu_B$

Test type: Mann-Whitney U test (one-tailed)

Test result: $p = 0.0021 \mid U = 104 \mid z = 2.86$

Accept H_{g,1}

Thus, the null hypothesis is rejected at a confidence level of 95% and our experiment provides evidence suggesting that H_g is valid. Additionally, we calculate the effect size according to Cohen’s *d* [Coh92].

Effect size: $d = 1.67$

Strong effect

According to the definition of Cohen’s *d*, values of $d > 0.8$ indicate a strong effect size. That is, we can observe a strong effect among the groups of our experiment.

7.2.3.4 Debriefing Questionnaire

In the debriefing questionnaire, we asked questions in two categories. First, there were questions about the tasks:

- How well did you understand the tasks?
- How difficult did you perceive the tasks?
- How do you estimate the quality of your results (high flexibility)?

Second, there were questions about modeling for flexibility. They targeted at observing cases in which participants recognized missing flexibility or changed their design for better flexibility.

- During architecture design, I changed my solutions when I recognized that the flexibility is insufficient (group A and group B)
- After modeling the change impact, I would have liked to change my architecture solutions as I recognized better possibilities (group A) /

After reasoning about / modeling the change impact, I changed my architecture solutions as I recognized better possibilities (group B)

- Modeling the change impact during architecture design would have helped me to come up with a better solution (group A) / Modeling the change impact during architecture design helped me to come up with a better solution (group B)

All questions were to be answered on a five-point Likert scale (1=Not Good / Easy / Low Quality / Fully Disagree .. 5=Good / Difficult / High Quality / Fully Agree). The original questionnaires can be found in Appendix B.

	<i>How well did you understand the tasks?</i> (5 = good)	<i>How difficult did you perceive the tasks?</i> (5 = difficult)	<i>How do you estimate the quality of your results (high flexibility)?</i> (5 = high)
Group A (n=8)	$\mu=3.50; \sigma=1.20$	$\mu=2.75; \sigma=1.04$	$\mu=3.00; \sigma=0,76$
Group B (n=9)	$\mu=3.56; \sigma=0.73$	$\mu=2.89; \sigma=0.93$	$\mu=3.33; \sigma=0.50$

Table 16: Debriefing questionnaire: Results on task-related questions

Table 16 depicts the average results of the task-related questions per group. We can observe the following aspects:

- There is nearly no difference in answers between groups A and B; that is they did not have a different perception of difficulty of tasks resulting from the experiment design. Additionally, participants of groups A and B come to a similar estimation of their quality of results, with a slightly higher value for group B.
- Participants indicate that they understood the tasks quite well ($\mu=3.53; \sigma=0.94$).
- Participants indicate that they did not perceive the tasks to be too difficult ($\mu=2.82; \sigma=0.95$).
- Participants estimate that they produced quite good results ($\mu=3.18; \sigma =0.64$).

	<i>During architecture design, I changed my solutions when I recognized that the flexibility is insufficient</i> (5 = Fully Agree)	<i>After reasoning about I modeling the change impact, I (would have liked to) changed my architecture solutions as I recognized better possibilities</i> (5 = Fully Agree)	<i>Modeling the change impact during architecture design (would have) helped me to come up with a better solution</i> (5 = Fully Agree)
Group A (n=8)	$\mu=3.38; \sigma=1.30$	$\mu=2.75; \sigma=1.16$	$\mu=4.00; \sigma=0.76$
Group B (n=9)	$\mu=2.56; \sigma=1.24$	$\mu=2.11; \sigma=1.27$	$\mu=3.11; \sigma=1.27$

Table 17: Debriefing questionnaire: Results on flexibility-related questions

Table 17 depicts the average results of the flexibility-related questions per group. We can observe the following aspects:

- For all questions there is a visible difference between group A and group B.
- In particular the last two questions are interesting. Group A indicated that they would have liked to change the architecture after modeling change impact more strongly than group B indicated that they did change after modeling change impact. Additionally, group A strongly indicated that modeling the change impact during architecture design would have helped them whereas there was weaker indication that group B perceived it as helpful.

7.2.4 Observations and Discussion

In this section, we describe observations on the experiment's conduction and results and on feedback from the participants. Further we discuss the potential practical benefits of the results shown.

Observations about participants

Flexibility as a quality attribute is not easy to understand. It always comes with indirections and in particular in the experiment the participants had to change the architecture in order to be well prepared for further changes. This led to the situation that some participants did not separate these levels and already described the final system, which was not helpful for the evaluation. One more observation about the participants is that they sometimes do not have enough knowledge about architecting and software development in general. They produced architectural diagrams, which made clear that they did not know what they abstracted from.

Nearly all participants were able to use the change impact notation although it was new to them and they produced quite good results. Additionally, they mentioned in the debriefing questionnaire that the tasks were not too difficult, including the drawing of the change impact views. This supports our hypothesis H_{11} (claiming the applicability that practitioners can describe flexibility solutions with change impact views). Even students with limited experience in architecting were able to use the change impact view notation. This makes us confident that it is also easy to use for experienced practitioners in architecting.

Feedback from participants

In general, there was not much feedback on the experiment in the comment section or also verbally. Interesting feedback was that some participants were influenced by knowing the architecture under design very well. As they knew all the rationales for design decisions, they tried to preserve the qualities of the system as demanded before. This, however, was not explicitly required in the task and might have led them to come up with less flexibility than would have been possible. Another feedback was that some students saw particular architectural aspects not as architectural but rather on the design level. However, as it was necessary to achieve a credible solution, such aspects had to be considered as architecturally important.

Potential of the method

The experiment covered only one contribution of this thesis which is expected to improve flexibility, namely the explicit modeling of change impact. The other main contributions are explicit support in designing for flexibility and tool-supported feedback on the currently achieved flexibility. It can be expected that even better results can be achieved if the different contributions of the thesis are combined.

Model for calculating benefits

Finally, we describe a simple model based on our experiment results which, with the help of some assumptions, calculates potential benefits of improved flexibility in practice. We assume a similar difference of flexibility between when using explicit change impact modeling (μ_B) and when not using it (μ_A) as observed in the experiment.

$$\mu_A - \mu_B = 0.27$$

In Section 7.2.2.1, we stated that, beyond the statistical significance, a difference of more than 0.1 on the flexibility scale $[0, 1]$ is expected. With the observed difference of 0.27, our expectation is even exceeded.

Assumptions:

- System with 1.000.000 LOC
- Flexibility of 0 defined for changing more than 10% of LOC

Conservatively, we assume that in such a large system with other competing quality requirements we achieve an improvement of 0.1 on the flexibility scale, which means we have to touch 1% less lines of code for

a specific change, which might mean to change 10.000 lines of code less in case of change when using the explicit change impact modeling.

7.2.5 Threats to Validity

In this section, we describe threats to validity of the experimental results and derived conclusions as well as which actions we applied to keep the threats small. We organize the threats in the following categories: Construct Validity, Internal Validity, External Validity, Conclusion Validity (see [WRH+00]).

7.2.5.1 Construct Validity

Construct validity is the degree to which the settings of the experiment in terms of the dependent and independent variables reflect the goal of the experiment. The following potential threats were identified:

Construction of the experiment:

- The input architecture to be changed in the experiment tasks was not deliberately designed to be best suited to test architecture design methods. Rather, it was taken as the students had designed it in their practical course. The current status after iteration 1 was taken in order to minimize description input and the time for understanding the architecture.
- The participants had to change an existing architecture. This might have reduced the perceived degree of freedom for architecture design as participants had to explicitly change existing design decisions instead of making them on the green field. In particular, the participants knew the design decisions and the rationales behind quite well and might not have changed certain aspects due to the knowledge of resulting tradeoffs or violations of other important aspects.
- The input architecture might also have some impact in the sense that it is not so far from being flexible for one or two of the scenarios. That is, the participants might have recognized this and seen this level of flexibility as sufficient, not improving the flexibility any more as it would have been possible.
- The experiment focused the tasks fully on the improvement of one quality attribute (namely flexibility), without considering the effect on other quality attributes. This might lead to different interpretations of the importance of tradeoffs for the participants.
- The selection of the flexibility scenarios was explicitly constructed to be well achievable with the input architecture. The flexibility scenarios are not designed for coverage of certain types of scenarios as introduced in this thesis. However, all the scenarios are representative of

practice and adapted from scenarios found in other projects with industry.

Construction of the measurement:

- The mathematical derivation of flexibility results from the participants' results requires complete adherence to the notation and estimation of size numbers by the participants. As nearly all participants did not deliver the complete data, the evaluation was approximated with an expert estimation.
- The expert estimation was done via one indirection: Judging the flexibility on a 5-point Likert scale, which was aligned with our flexibility metric. However, this Likert scale is a discrete scale which is rather coarse-grained. For a single estimation, it might lead to a deviation, but an expert cannot make a more precise estimation based on the degree of precision of architecture results provided by the students.
- The measurement of flexibility requires interpretation of the ideas of the participants of how to achieve flexibility. However, the evaluating person is an experienced architect with knowledge about all typical flexibility concepts as used by the students and also knowledge about the consequences on the implementation.
- The measurement of flexibility required filtering out inappropriate architectural solutions, which could not be used as a basis for flexibility measuring (independent of the achieved flexibility, the solutions were generally not appropriate). This required an additional step of estimating appropriateness, which is another potential source of wrong expert estimations.
- The flexibility results for the three scenarios were mixed for evaluation, in a pool for group A and group B respectively. This leads to a comparison that does not exactly compare the same scenarios for group A and group B directly. The individual result sets were too small for statistical testing.
- The flexibility measurement and the estimations were done by the method owner who had also designed the experiment. To mitigate the risk of wrong estimations, the actions as described in Section 7.2.3.1 were taken.

7.2.5.2 Internal Validity

Internal validity is the degree to which independent variables have an impact on dependent variables. The following threats to internal validity have been identified:

- Assignment of participants to the experiment groups can lead to selection effects. As the number of participants is quite low, such an effect can have larger effects than in big studies. The assignment of our

participants to groups A and B was done randomly. With the help of the briefing questionnaire we checked with their experience and did not find a significant difference.

- We did not conduct a cross-design experiment in order to mitigate influence of skills: The learning effect after having conducted the tasks with explicit change impact modeling is expected so strong that such an experiment design does not appear feasible.
- A further selection effect could be based on the different degree of knowledge of the architecture to be changed. The students acting as architects in the course could be expected to know it better. However, we checked this and found that on the one hand three architects were assigned to group B and only one to group A. Nevertheless, also two of the architects in group B did not produce appropriate architectural solutions and so their solutions could not be evaluated completely.
- The participants did not all spend the same time on conducting the tasks. No participant exceeded the allotted time, but some did it in roughly half the time.

7.2.5.3 External Validity

External validity is the degree to which the results of the experiment can be transferred to other people and to changed environmental settings. The generalizability of the results is limited due to the following facts:

- The system, in the context of which the experiment was conducted, is still quite small and does not reach the size of typical information systems in industry.
- The participants in the experiment are all students at the end of their studies of computer science / software engineering. That is, they are no experienced developers and in particular no software architects with a history of practical experience.
- The tasks to optimize the architecture of a software system only with respect to a single quality attribute, in our case flexibility, is not representative. In practice, there are always competing requirements and the architect has to find appropriate solutions and tradeoffs. However, the task to optimize flexibility and the input scenarios are highly realistic and the scenarios are slightly adapted ones from other industrial projects.

7.2.5.4 Conclusion Validity

Conclusion validity is the degree to which the concluded results of the experiment reflect the measured effects and are not corrupted by inappropriate or insufficient statistical methods.

- The resulting flexibility values for groups A and B are not normally distributed. Thus, we selected the Mann-Whitney-U test instead of a t-test, which is a bit less conservative.
- The sample size of the flexibility values in both groups A and B is quite small. However, the sample sizes are similar and our statistical test shows a significant difference.

7.3 Project Experiences

In Section 7.2, we described the isolated evaluation of the effect of change impact views during architecture design in an experiment. Additionally, we applied parts of our method in several projects with industrial customers of Fraunhofer IESE. Selected aspects of the contributions of this thesis were applied in order to address the respective project goals of the customer. While in our experiment the validation focus is on the measurement of flexibility (R.I3), we mainly applied the conceptual model (R.I1) and the constructive guidance (R.I2) in the projects with industrial customers.

As described in Figure 2, we used earlier projects for the identification and confirmation of the problems addressed in this thesis. Later on, we applied first versions of the contributions in projects and refined the contributions based on the experiences we made. In this section, we report on three recent projects in which our contributions were partially applied. Due to the confidentiality and non-disclosure agreements with the customers, we report anonymously on these projects and call them project A, project B, and project C. The projects are described following a uniform structure: First, we describe the context and the goals of the respective project. Then, we list the contributions of this thesis which were applied in the project. Finally, we report on the results and lessons learned in the project with respect to the contributions of this thesis.

Summary of experiences, support for hypotheses

Although we applied only parts of the approach in the projects, we perceive that the approach strongly supports more systematic working with the quality attribute flexibility. In addition, the stakeholders on customer side, both management stakeholders and architects, confirmed the usefulness of conceptual and method parts of our approach. This in particular supports our hypothesis H_3 which claims that the conceptual model supports stakeholders in understanding the quality attribute flexibility.

Additionally, we experienced the separation of business-logic-agnostic and business-logic-specific architecture in a positive way. It turned out to be a strong enabler for understanding flexibility and for designing for it. This supports our hypothesis H_4 which claims the effectiveness of our design support resulting in better flexibility.

Further, we experienced collecting flexibility requirements, supported by our characterization of flexibility, in a positive way. In particular the explicit guidance towards typical changes and the questions to characterize potential changes in detail helped. This supports our hypothesis H_5 which claims that our guidelines improve the elicitation of flexibility requirements.

Finally, we also learned about the effort to be spent on flexibility-specific design activities. There we found that typically architects could make well-founded decisions without spending much additional effort. In particular compared to expensive later changes, the investments into a flexible architecture should pay off, which supports our hypothesis H_6 claiming efficiency in the sense that the additional investments into design for flexibility are low compared to the avoided costs for changing an inflexible system.

7.3.1 Project A

Project context and goals

- SOA-based information system, existing system
- Goal: need for integration with completely different backend system
- Many new flexibility scenarios to be addressed, but also other quality attributes
- Flexibility scenarios mainly with focus on integration with external systems and the change of business processes in a data-intensive application
- Goal: architecture redesign to match new requirements and subsequent implementation
- Setting: Coaching of customer organization architects with respect to architecture design methods; customer organization architects applied the methods

Contributions of this thesis applied

- Guidelines for the elicitation of flexibility requirements
- Separation of BLA and BLS aspects
- Guidelines and heuristics for architecture design addressing flexibility scenarios, in particular the separation of *Infrastructure*, *Template*, *Business* elements and their usage in design

Results and lessons learned

- Detailed specification and characterization of flexibility requirements
- Elicitation of flexibility requirements was guided by the experience of historical change requests to the system
- Classification of flexibility requirements according to:
 - *What has to change?*
 - *What is impacted by the change from an architecture perspective?*
 - *Which concrete business logic aspects are related to the change?*
- Guidelines considerably helped architects to come up with flexibility requirements and ask the right questions to stakeholders
- Heuristics and guidelines for design with separated *Infrastructure*, *Template*, *Business* elements were reported to be very helpful by the architects
- Applying the approach does not lead to much additional effort

7.3.2 Project B

Project context and goals

- SOA-based information system, system's implementation nearly finished
- Goal: Evaluate whether the built-in configuration mechanisms are appropriate in terms of expressiveness and flexibility or whether the usage of a commercial rule engine would be better
- Fraunhofer IESE as independent reviewer for customer, project conducted by method owner of flexibility engineering approach

Contributions of this thesis applied

- Guidelines for the elicitation of flexibility requirements
- Conceptual model showing the relationship of flexibility to requirements, architecture, and implementation
- Separation of BLA and BLS aspects
- Change impact views for the visualization of expected change impacts: used for comparison of change impacts between individual configuration framework and a rule engine solution

Results and lessons learned

- Flexibility conceptual model and in particular BLA / BLS are applicable to business rule systems where it had not been applied before (defining business rules for higher flexibility also means to get the mapping of business logic to rules right)
- BLS-level is necessary to be able to really check whether flexibility is given
- Change impact views (slightly simplified) perceived very positively also by business stakeholders who understood the differences between the two compared solution approaches
- Results about expected flexibility of both solution approaches: very similar, the difference is more in the external support of business rule management systems (usability, testing of rules)

7.3.3 Project C

Project context and goals

- Information system; existing system, which can be only delivered as a monolithic block containing all functionality although it might not be completely needed
- Goal: Modularize the system in order to provide more independent services which can be used in external systems and business processes. The business rationale is to increase the market (smaller customers, partnering) and to be able to compose new products
- Project conducted as consulting project of Fraunhofer IESE, mainly aiming at the identification of modularization opportunities and at the definition of a new target architecture with more flexibility

Contributions of this thesis applied

- Guidelines for the elicitation of flexibility requirements
- Conceptual model showing the relationship of flexibility to requirements, architecture, and implementation
- Separation of BLA and BLS aspects
- Enhanced architecture design process for flexibility, with a focus on *Existing Artifacts Analysis*

Results and lessons learned

- Guidelines for elicitation of flexibility requirements applicable and helpful: 9 key flexibility scenarios elicited in 6 hours with key stakeholders of the system
- New, project-specific classification for flexibility requirements introduced: Who conducts the changes? (company itself, partnering companies, outsourcing companies)
- BLS-level is critical for formulating adequate flexibility scenarios
- Integrated analysis of functionality, processes, and data have to be considered for flexibility improvement

8 Summary and Outlook

*“It is not our duty to predict the future,
but to be prepared for it.”*
Pericles

This section concludes the thesis. First, we summarize the results and contributions of the thesis in Section 8.1. Then, we discuss limitations, and sketch future activities beyond the thesis in Section 8.2. Finally, we close with some concluding remarks in Section 8.3.

8.1 Results and Contributions

Thesis goals	This thesis originates in the discovery of flexibility problems in SOA-based information systems. Although SOA is widely known for its flexibility potential and many practitioners even expect inherent flexibility, SOA-based systems in practice are often not flexible enough. We set the goal to support architects in building flexible SOA-systems by systematically exploiting the flexibility potential of SOA architecture mechanisms.
True flexibility	We discovered that a key reason for missing flexibility is the lack of alignment of SOA architecture mechanisms with business logic mapping to these architecture mechanisms. Thus, we define the term <i>True Flexibility</i> , which denotes that architecture mechanisms and business logic mapping are aligned in a way that the resulting flexibility potential matches the flexibility requirements.
Constructive support missing	Analyzing related work around flexibility and architecting shows that even for information systems in general (without a focus on SOA) there is a lack of constructive support for designing flexible architectures. While there are several methods for analyzing architectures with respect to flexibility or maintainability, the constructive support is restricted to the provision of architectural mechanisms that can help to achieve flexibility.
General and SOA-specific contributions	Consequently, we split our contributions in a general part, which supports architects of any kind of software system, and a SOA-specific part. In order to focus our research activities, we first defined research direc-

tions which were considered to contribute to the achievement of our goals. With the help of analyzing related work, we came up with concrete research challenges in the areas of all the research directions. In the following, we summarize the key contributions of this thesis.

Conceptual Model

Characterization of flexibility

The underlying foundation of the methodical contributions is a detailed characterization of flexibility as a quality attribute of software systems. We depict what flexibility means in terms of building in flexibility during system design and exploiting it in later life-cycle phases of the systems when changes have to be made. We characterize and define flexibility in a way spanning software engineering disciplines, from flexibility requirements over the role of architecture to the role of implementation. At architectural level, we elaborate how flexibility can be achieved and which information in architecture meta-models is flexibility-relevant. We define a metric for flexibility which measures the expected change impact with respect to flexibility scenarios and which can be aggregated to the overall architecture level and multiple flexibility scenarios. These characterizations of flexibility are summarized in a **conceptual model**, which is represented in four views. It can serve architects and other stakeholders in software development as a map of guidance for the understanding of flexibility as a quality attribute.

Methodical Contribution

A key goal of this thesis is to support architects in constructively achieving flexibility. Our approach is to enhance existing architecture design methods. We provide a conceptual plugin into architecture design methods which builds on flexibility-specifics and uses these to give architects more guidance. It consists of mainly two parts: 1) guidelines and heuristics for architecture decision making and 2) continuous measuring of flexibility during architecture design with direct feedback for architects.

Architecture decisions for flexibility

For the constructive part, we outline a **design process for flexibility**, extending activities of existing architecture design methods. For the individual activities, we give flexibility-specific guidelines and heuristics, in particular with respect to the intertwining of selecting architecture mechanisms and defining business logic mappings.

Continuous flexibility measurement

A further contribution for architects in designing flexible architectures is that we introduce a possibility to **continuously measure flexibility** and give **near-real-time feedback** to architects. While the evaluation of achieved flexibility is typically a manual task which, if at all, is done after architecture design we automate the measurement of flexibility. The key

idea behind this is that architects can immediately see the impact of their architecture decisions on flexibility, allowing quick revisions of suboptimal design decisions. The typical way of documenting software architecture and requirements does not allow for this measurement. Thus, we introduce an extension to the architecture model, the so called **change impact view**. This view is modeled in a lightweight way by the software architect when he designs the architecture. This does not add much overhead as architects should make the considerations they make persistent in the model anyway. Then, flexibility can be automatically measured according to our metric and the results can be provided to the architect.

Tool Support for automated measurement

Enterprise
Architect
AddIn

The **computation and representation of flexibility metrics** is integrated as a proof-of-concept in a widely-used **architecture modeling tool**, Enterprise Architect. With this contribution, we allow architects to model their architectures with optimal support of the flexibility-specific enhancements. The underlying architecture meta-model is realized in the tool and the architect can very easily model change impact views. Graphical representations give in-detail and overview information on the flexibility metrics currently achieved.

SOA-specific Support for Flexibility

SOA chal-
lenges, solu-
tions, and
technologies

In addition to these generally applicable contributions, we also make a contribution specific to **flexibility in SOA-based systems**. We collect **typical flexibility requirements in SOA**-based software systems as guidance for the elicitation of flexibility requirements. Additionally, we analyze and **package architectural mechanisms used in SOA** with respect to their support for flexibility. We elaborate these architectural mechanisms in a uniform way with their typical architectural principles and design decisions. Further, we sketch which architectural elements are typically found, how they are related to each other, and how this contributes to flexibility. A specific focus is on making the relationship of architecture mechanisms and business logic mapping explicit. Finally, we give an overview of how **SOA-technologies** realize the architecture mechanisms and thus can support architects in achieving flexibility. The contribution is an explicit description of architectural aspects around flexibility in SOA which guides architects during architecture design tasks. This contribution leads us back to our initial starting point of the thesis, namely supporting architects in building flexible SOA-based software systems.

Validation

We validated parts of our contributions in a controlled experiment and had first experiences with the method in projects with customers from industry.

Controlled experiment

In a **controlled experiment** in the context of a practical course at TU Kaiserslautern, we analyzed how the **explicit modeling of change impact views** during architecture design impacts the flexibility of the resulting system. We found that the modeling of change impact views alone, even without tool-supported automated measurement, leads to a strong improvement of flexibility compared to a control group. The measured effect was statistically significant and had a strong effect size. This experiment gives evidence that architects, supported by our methodical enhancement, can design architectures that are more flexible; thus less effort for changes has to be spent.

8.2 Limitations and Future Work

This section sketches limitations of the approach and validation contributed in this thesis and proposes research directions for future work. We align this section with the areas of contributions of this thesis and elaborate additional research directions in terms of practical applicability.

Enhanced conceptual model and flexibility metrics

Cover other activities

Our conceptual model and flexibility metrics concentrate on the effort for changing the implementation only. The flexibility metrics could be extended towards covering other software engineering activities necessary in line with software changes, like build and deployment.

Combination with other metrics

In the flexibility metric, only the size of the impacted implementation is considered, but not other properties like the readability of the source code. Such properties, e.g. expressed by metrics like code complexity, could be combined with the flexibility metric in order to give a holistic picture on the expected change effort.

Formal ROI model for flexibility

Our flexibility metric covers only the effort for conducting anticipated changes, but it does not take into account how much the building in of this flexibility really costs. By extending the metric and finding a good way of measuring this cost, the return on investment into flexibility can be formalized and calculated.

Constructively guiding all stakeholders towards flexibility

Better integration with requirements engineering

So far, the approach of this thesis mainly supports architects and to some extent requirements engineers. However, the guidance for requirements engineers is not integrated with requirements engineering approaches but rather describes relevant questions and areas of requirements elicitation. More integration of the overall approach with requirements engineering is desirable. This would also involve a concept for traceability among requirements that are realized and requirements that concern flexibility and only prepare future changes. This could help to identify change impacts and serve as a first automated approximation of change impact views, assumed that full traceability of requirements to realizing architectural elements exists.

Exploitation of flexibility

Our approach covers only the construction of flexibility, but not how to conduct concrete change requests. More support for architects and in particular developers is desirable for making best use of the flexibility potential built into a software system. Only then, flexibility is supported over the full lifecycle of a software system.

Tool support

Support for modeling change impact views

Currently, modeling and maintaining change impact views is a completely manual task for architects. By the means of formalized heuristics, tool-supported creation of change impact views could be possible. With the help of name-based heuristics, requirements traceability, or knowledge about relationships between template and business elements in the architecture model, likely change impacts could be identified and suggested to the architect.

Support for architectural decisions

The constructive process of making architectural decisions towards a flexible architecture is not tool-supported at the moment. An interesting research direction would be to find out how business logic mapping to architectural mechanisms could be tool-supported with adequate heuristics.

Automated extraction of facts about existing implementation

In practice, architects are often concerned with existing systems and have to change them in order to have more flexibility in the future. Then, far more concrete information on existing implementation artifacts is available and can be used to make the calculations of flexibility more precise. In particular, the implementation size could be automatically extracted and integrated into the architecture model.

SOA and other development paradigms

Business rules and event-driven architecture

In this thesis, we analyzed the architectural mechanisms of SOA with respect to their contribution to flexibility. Looking at architectural mechanisms like patterns in general reveals that there are too many to analyze all of them with respect to flexibility. However, there are other interesting paradigms like business rules or event-driven architecture which also promise strong support for flexibility. They should be analyzed in more depth to give architects more guidance about their flexibility potential.

Validation

In this thesis, we outline several hypotheses which could not be evaluated in the context of the thesis. Thus, more empirical work around the approach is desirable.

Flexibility metric

Although our flexibility metric is comprehensively constructed in a way that directly covers the size of impacted implementation, there should be more evidence that a better flexibility value measured on the architecture also leads to lower implementation effort (H_2). Thus, an experiment should be conducted, which also covers the realization of change requests, for which the architecture was made flexible. This experiment could be set up in a way that different architectural solutions are provided. Then, first a measurement of flexibility according to our metric could be conducted and second an implementation of the changes could be done, observing the effort spent and the real change impact.

Design for flexibility requirements vs. "good design"

Our approach builds on the usage of flexibility scenarios as a basis for measuring flexibility since an architecture cannot be flexible with respect to all changes. A very interesting and challenging research question is how big, on average, the gap is between explicit design for concrete flexibility scenarios on the one hand and following the principles of good design on the other hand (see Section 4.1.4).

Impact of tool support

So far, we have not included the tool-support in evaluation activities. However, in particular the hypotheses H_8 and H_9 are closely related to each other. Thus, it should be evaluated which effects on flexibility occur when the tool support is used. Particularly in large architecture models we expect that tool-support helps to control the complexity and thus might get stronger influence.

Practical applicability and situations in projects

Transfer to industry

In order to fully apply our approach in practice, organizations need a certain maturity in architecting. At least architecture models in a modeling tool are necessary in order to apply automated measurement of

flexibility. The guidelines for construction of flexibility can also be applied in rather informal settings. For sustainable transfer of the approach into industrial organizations, an introduction concept should be elaborated.

Project constellations and impact on flexibility

Not all software projects are in a situation that building in flexibility is desired by all organizations involved. Often, there is the situation that a company develops a software system but is not involved in the maintenance. Then, flexibility is mainly a cost driver but does not provide immediate benefit for the developing company. As flexibility is hard to measure by the customer, it is then often neglected. There are more situations like this and thus it would be worthwhile to characterize the different constellations of organizations and their stakes in flexibility during the lifecycle of a software system. This characterization could serve as a guideline for companies involved to become aware of their responsibilities and opportunities with respect to flexibility.

8.3 Concluding Remarks

The research conducted in this thesis follows the principles of applied research. In projects with industrial customers we identified the problem that SOA-based information systems are not flexible enough and not as flexible as expected. Then, we analyzed the reasons for the missing flexibility and reviewed the state-of-the-art in the identified research directions. We separated our contributions into SOA-specific and general architecture contributions and came up with the key idea to support architects with both, constructive guidelines and tool-supported near-real-time analytics of flexibility. We elaborated the different solution components and implemented a tool prototype. Finally, we evaluated parts of our approach in an experiment and in industrial projects.

With our contributions, we showed how to holistically cover a quality attribute, in our case flexibility. We provide a conceptual model, guidance for the elicitation of the concrete flexibility requirements, constructive and analytical guidance during architecture design, and tool-support. In future research, other quality attributes should be covered in a similar way. Then, integration on the same architecture model becomes possible and brings us closer to the vision of architecture-centric engineering, with all individual and tradeoff analyses being possible at the architecture level.

We will further apply our approach in projects with industrial customers and collect more experience, which will help to focus our future improvements of the approach. We are confident that we can improve the flexibility of large information systems and help companies to realize critical changes faster, resulting in lower costs for software maintenance and in higher competitiveness.

References

- [AA06] Ali Arsanjani, Abdul Allam. *Service-Oriented Modeling and Architecture for Realization of an SOA*. in Proceedings of the IEEE International Conference on Services Computing: IEEE Computer Society, pp. 521, 2006.
- [ABB+02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, Jörg Zettel. *Component-based product line engineering with UML*. Addison-Wesley Longman Publishing Co., Inc., 2002.
- [ABK10] Pekka Abrahamsson, Muhammad Ali Babar, Philippe Kruchten. *Agility and Architecture: Can They Coexist?*. IEEE Software 27(2): 16-22, 2010.
- [ADE+09] Sebastian Adam, Jörg Dörr, Michael Eisenbarth, Anne Gross. *Using Task-oriented Requirements Engineering in Different Domains – Experiences with Application in Research and Industry*. Proceedings of the IEEE International Requirements Engineering Conference. IEEE Computer Society, 2009.
- [AFL+05] Bettina Anders, Jörg Fellmann, Mikael Lindvall, Ioana Rus. *Experimenting with Software Architecture Flexibility Using an Implementation of the Tactical Separation Assisted Flight Environment*. SEW 2005: 275-284, 2005.
- [AGA+08] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Gariapathy, and K. Holley. *SOMA: a method for developing service-oriented solutions*. IBM Syst. J., 47(3):377-396, 2008.
- [AH06] Ali Arsanjani, Kerrie Holley. *The Service Integration Maturity Model: Achieving Flexibility in the Transformation to SOA*. Proceedings of the IEEE International Conference on Services Computing (SCC06), 2006.
- [Ama11] Amadeus. Airline IT Solutions – Full Altea Suite. http://www.amadeus.com/airlineIT/solutions/sol_1altea_1suite_1full.html - last visited 18.12.2011 -
- [ANT10] Sebastian Adam, Matthias Naab, Marcus Trapp. *A Service-Oriented View on Business Processes and Supporting Applications*. Enterprise, Business-Process and Information Systems Modeling - 11th International Workshop, BPMDS, 2010.
- [ANT+11] Sebastian Adam, Matthias Naab, Marcus Trapp, Steffen Olbrich. *Conceptual Model for Service Oriented Architecture (SOA) and Service Oriented Engineering (SOE)*. IESE-Report 020.11/E.
- [AZE+07a] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, Kishore Channabasavaiah. *S3: A Service-Oriented Reference Architecture*. IEEE IT Professional. May/June 2007 (vol. 9 no. 3), 2007.
- [AZE+07b] Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, Kishore Channabasavaiah. *Design an SOA solution using a reference architecture*. <http://www.ibm.com/developerworks/library/ar-archtemp/> - last visited 31.10.2011 -
- [BAA10] Hans Christian Benestad, Bente Anda, Erik Arisholm. *Understanding cost drivers of software evolution: a quantitative and qualitative investigation of change effort in two evolving software systems*. Empirical Software Engineering 15(2):166-203, 2010.

- [Bah05] Rami K. Bahsoon. *Evaluating Architectural Stability with Real Options Theory*, PhD Thesis, University of London, 2005.
- [Bal09] Sebastian Ballhausen. *Erfolgsfaktoren für eine business- & wertorientierte IT*. http://www.boydak.biz/cms/fileadmin/Webmaster/pdfFiles/Artikel_2009/2009-02-16-automotive-it.pdf
- last visited 03.08.2011 –
- [Bar03] Barbacci, M. *Software Quality Attributes and Architecture Tradeoffs*. Software Engineering Institute, Carnegie Mellon University. Pittsburgh, PA, 2003
- [BB99] PerOlof Bengtsson, Jan Bosch. *Architecture Level Prediction of Software Maintenance*. CSMR 1999: 139-147, 1999.
- [BB00] PerOlof Bengtsson, Jan Bosch. *An experiment on creating scenario profiles for software change*. Ann. Software Eng. 9: 59-78, 2000.
- [BB01] Jan Bosch, PerOlof Bengtsson. *Assessing Optimal Software Architecture Maintainability*. Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR). IEEE Computer Society, 2001.
- [BBN07] Felix Bachmann, Len Bass, Robert Nord. *Modifiability Tactics*. CMU/SEI-2007-TR-002, 2007.
- [BC10] Hongyu Pei Breivold, Ivica Crnkovic: *A Systematic Review on Architecting for Software Evolvability*. Australian Software Engineering Conference 2010: 13-22, 2010.
- [BCE08] Hongyu Pei Breivold, Ivica Crnkovic, Peter J. Eriksson. *Analyzing Software Evolvability*. COMPSAC 2008: 327-330, 2008.
- [BCK03] Len Bass, Paul Clements, Rick Kazmann. *Software Architecture in Practice, 2nd Edition*. Addison-Wesley Longman, 2003.
- [BD88] V. R. Basili, H. D. Rombach: *The TAME project. Towards improvement-oriented software environments*. In: IEEE Transactions on Software Engineering. Bd. 14, Nr. 6, S. 758–773, 1988.
- [BDP06] Manfred Broy, Florian Deissenböck, Markus Pizka. *Demystifying Maintainability*. Proceedings of the 2006 International workshop on Software quality (WoSQ 06), 2006.
- [BE03] Rami Bahsoon, Wolfgang Emmerich. *Evaluating Software Architectures: Development Stability and Evolution*. Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications, Tunis, Tunisia. (pp. 47 - 56), 2003.
- [BE04] Rami Bahsoon, Wolfgang Emmerich. *Evaluating Architectural Stability with Real Options Theory*. ICSM 2004: 443-447, 2004.
- [BE06] Rami Bahsoon, Wolfgang Emmerich. *Requirements for Evaluating Architectural Stability*. International Conference on Computer Systems and Applications 2006: 1143-1146, 2006.
- [Ben02] PerOlof Bentsson. *Architecture-Level Modifiability Analysis*. Dissertation, Department of Software Engineering and Computer Science, Blekinge Institute of Technology, Sweden, 2002.
- [BHS+08] Hans-Jörg Beyer, Dirk Hein, Clemens Schitter, Jens Knodel, Dirk Muthig, Matthias Naab. *Introducing Architecture-Centric Reuse into a Small Development Organization*. International Conference on Software Reengineering (ICSR), 2008.

- [BKL+95] Mario Barbacci, Mark Klein, Thomas Longstaff, and Charles Weinstock. *Quality Attributes*. Technical Report CMU/SEI-95-TR-021 ESC-TR-95-021.. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. 1995.
- [BKR07] Steffen Becker, Heiko Koziolok, Ralf Reussner. *Model-Based performance prediction with the palladio component model*. WOSP 2007: 54-65, 2007.
- [BKR09] Steffen Becker, Heiko Koziolok, Ralf Reussner. *The Palladio component model for model-driven performance prediction*. Journal of Systems and Software 82(1): 3-22, 2009.
- [Bla01] Sue Black. *Computing ripple effect for software maintenance*. J. Softw. Maint. Evol.: Res. Pract. 2001; 13:263–279, 2001.
- [BLB+00] PerOlof Bengtsson, Nico H. Lassing, Jan Bosch, Hans van Vliet. *Analyzing Software Architectures for Modifiability*. 2000.
- [BLB+04] PerOlof Bengtsson, Nico H. Lassing, Jan Bosch, Hans van Vliet. *Architecture-level modifiability analysis (ALMA)*. Journal of Systems and Software 69(1-2): 129-147, 2004.
- [BLM+11] Philip Bianco, Grace A. Lewis, Paulo Merson, Soumya Simanta. *Architecting Service-Oriented Systems*. SEI Technical Note, CMU/SEI-2011-TN-008, 2011.
- [BMR97] Frank Buschmann, Renie Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. Wiley, 1997.
- [Boo06] Grady Booch. *On Architecture*. IEEE Software 23(2): 16-18, 2006
- [Boo07] Grady Booch. *The Economics of Architecture-First*. IEEE Software 24(5): 18-20, 2007.
- [Bos00] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley Longman, 2000.
- [Bos10] Jan Bosch. *Architecture in the Age of Compositionality*. Keynote at European Conference on Software Architecture (ECSA) 2010, 2010,.
- [BPEL] OASIS. *Web Services Business Process Execution Language (WSBPEL)*. www.oasis-open.org/committees/wsbpel
- last visited 19.12.2011 -
- [BS09] Caroline Buck, Markus Schärtel. *Architekturkonzepte mit Business Rules*. JavaSPEKTRUM 05/2009.
- [CAL+94] Don Coleman, Dan Ash, Bruce Lowther, Paul Oman. *Using Metrics to Evaluate Software System Maintainability*. IEEE Computer, 27(8), pp. 44-49, 1994.
- [Car12] Ralf Carbon. *Architecture-Centric Producibility Analysis*. PhD-Thesis, Fraunhofer IRB-Verlag, 2012.
- [CBB10] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord. *Documenting Software Architectures: Views and Beyond*, 2nd revised edition. Addison Wesley, 2010.
- [Cha04] David Chappell. *Enterprise Service Bus*. O'Reilly Media, 2004.
- [Che08] Betty H. Cheng et al. *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. In Software Engineering for Self-Adaptive Systems. Lecture Notes In Computer Science, Vol. 5525. Springer-Verlag, 2008.

- [CJM+08] Ralf Carbon, Gregor Johann, Dirk Muthig, Matthias Naab. *A Method for Collaborative Development of Systems of Systems in the Office Domain*. EDOC, 2008.
- [CKK01] Paul Clements, Rick Kazman, Mark Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison Wesley, 2001.
- [CK06] Patricia Costa, Thorsten Keuler. *Architectural Flexibility for Devices in the Virtual Office of the Future*. IESE-Report No. 016.06/E, 2006.
- [Cle10] Paul C. Clements: *Certified Software Architects*. IEEE Software 27(6): 6-8, 2010.
- [CN07] Paul Clements, Linda M. Northrop. *Software Product Lines : Practices and Patterns*. Addison Wesley, 2007.
- [CN10] Ralf Carbon, Matthias Naab. *Facilitating Evolution by Architectural Design for Flexibility and Buildability*. 2nd Workshop of GI Working Group „Long-living Software Systems (L2S2): Design for Future. Bad Honnef, Germany, 2010.
- [Coh92] Jacob Cohen. *A power primer*. Psychological Bulletin 112: 155–159, 1992.
- [CS09] Paul C. Clements, Mary Shaw. *“The Golden Age of Software Architecture” Revisited*. IEEE Software 26(4): 70-72, 2009.
- [Doe11] Jörg Dörr. *Elicitation of a Complete Set of Non-Functional Requirements*. PhD Thesis, Fraunhofer IRB Verlag, 2011.
- [Dij82] Edsger W. Dijkstra. *On the role of scientific thought*. In Dijkstra, Edsger W.. Selected writings on Computing: A Personal Perspective. New York, NY, USA: Springer-Verlag New York, Inc.. pp. 60–66, 1982.
- [DWP+07] Florian Deissenboeck, Stefan Wagner, Markus Pizka, Stefan Teuchert, Jean-Francois Girard. *An Activity-Based Quality Model for Maintainability*. ICSM 2007: 184-193, 2007.
- [EA11a] Sparx Systems. *Enterprise Architect: UML tools for Software Development and Modeling*.
- last visited 11.12.2011 -
- [EA11b] Sparx Systems. *Enterprise Architect: Automation and Scripts*.
- last visited 11.12.2011 -
- [EHH08] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus Voß, Johannes Willkomm. *Quasar Enterprise – Anwendungslandschaften service-orientiert gestalten*. dpunkt.verlag, 2008.
- [EKK+10] Michael Eisenbarth, Thorsten Keuler, Jens Knodel, Matthias Naab, Dominik Rost. *Fraunhofer DSSA*. IESE-Report No. 035.10/E, 2010.
- [EM06] Amnon H. Eden, Tom Mens. *Measuring Software Flexibility*. IEEE Software, Vol. 153, No. 3 (Jun. 2006), pp. 113–126. London, UK: The Institution of Engineering and Technology, 2006.
- [End04] Rainer Endl. *Regelbasierte Entwicklung betrieblicher Informationssysteme - Gestaltung flexibler Informationssysteme durch explizite Modellierung der Geschäftslogik*. Dissertation, Bern, 2004.
- [ENS07] Stefan Eicker, Anett Nagel, Peter M. Schuler. *Flexibilität im Geschäftsprozessmanagement-Kreislauf*. ICB-Research Report No.21, 2007.
- [ER03] Albert Endres, H. Dieter Rombach. *A Handbook of Software and Systems Engineering - Empirical Observations, Laws and Theories*. Addison-Wesley, 2003

- [Erl06] Thomas Erl. *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall, 2006.
- [EPIC] Electronic Privacy Information Center (EPIC). *EU-US Airline Passenger Data Disclosure*.
http://epic.org/privacy/intl/passenger_data.html
- last visited 18.12.2011 -
- [Fai10] George Fairbanks. *Just Enough Software Architecture – A Risk-Driven Approach*. Marshall & Brainerd, 2010.
- [Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2003.
- [Fri09] Uwe Friedrichsen. *Der Mythos Wiederverwendung: „Design für Wartung“ als eigentliches Ziel*. ObjektSpektrum 04/2009.
- [GAO09] David Garlan, Robert Allen, John Ockerbloom. *Architectural Mismatch: Why Reuse Is Still So Hard*. IEEE Software, Volume 26 Issue 4, July 2009.
- [GBD08] Vish Ganapathy, Melody Badgett, Jay DiMare. *Rethinking retailing with SOA - New levels of flexibility, agility and cost-efficiency*. IBM Institute for Business Value, 2008.
- [GBS01] Jilles van Gorp, Jan Bosch, Mikael Svahnberg. *On the Notion of Variability in Software Product Lines*. Working Conference on Software Architecture (WICSA) 2001: 45-54, 2001.
- [GBS+09] David Garlan, Jeffrey M. Barnes, Bradley R. Schmerl, Orieta Celiku. *Evolution styles: Foundations and tool support for software architecture evolution*. WICSA/ECSA 2009: 131-140, 2009.
- [Geb11] Michael Gebhart. *Qualitätsorientierter Entwurf von Anwendungsdiensten*. Dissertation, KIT Scientific Publishing, 2011.
- [GHJ94] Erich Gamma, Richard Helm, Raph. E. Johnson, John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, 1994.
- [GL05] Judith Gebauer, Fei Lee. *Towards an "Optimal" Level of Information System Flexibility - A Conceptual Model*. ECIS 2005: 373-380, 2005.
- [GL08] Judith Gebauer, Fei Lee. *Enterprise System Flexibility and Implementation Strategies: Aligning Theory with Evidence from a Case Study*. IS Management 25(1): 71-82, 2008.
- [Gor06] Ian Gorton. *Essential Software Architecture*. Springer, 2006.
- [GS94] David Garlan, Mary Shaw. *An Introduction to Software Architecture*. Advances in Software Engineering and Knowledge Engineering, Volume 1: World Scientific Publishing Company, 1994.
- [GS06] Judith Gebauer, Franz Schober. *Information System Flexibility and the Cost Efficiency of Business Processes*. Journal of the Association for Information Systems 7(3), 2006.
- [GS09] David Garlan, Bradley R. Schmerl. *Ævol: A tool for defining and planning architecture evolution*. ICSE 2009: 591-594, 2009.
- [HHV06] Andreas Hess, Bernhard Humm, Markus Voß. *Regeln für serviceorientierte Architekturen hoher Qualität*. Informatik-Spektrum Vol. 29, Springer-Verlag, 2006.

- [HKN+07] Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk, Alexander Ran, Pierre America. *A general model of software architecture design derived from five industrial approaches*. Journal of Systems and Software, Vol. 80, No. 1., pp. 106-126, 2007.
- [HNS99] Christine Hofmeister, Robert Nord, Dilip Soni. *Applied Software Architecture: A Practical Guide for Software Designers*. Addison-Wesley Longman, 1999.
- [IATA] International Air Transport Association (IATA).
www.iata.org
- last visited 18.12.2011 -
- [IBM06] IBM. *Service Oriented Architecture: Flexibility for Business*. Executive Brief. 2006.
<ftp://ftp.software.ibm.com/common/ssi/pm/xb/n/sme00235usen/SME00235USEN.PDF>
- last visited 23.12.2011 -
- [IBM11] IBM. *Maximize efficiency and flexibility with Cloud computing*.
<http://www-935.ibm.com/services/be/reducecosts/cloud>
- last visited 19.12.2011 -
- [IEEE90] IEEE. *Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*. 1990.
- [IEEE00] ANSI/IEEE 1471-2000. *Recommended Practice for Architecture Description of Software-Intensive Systems*. 2000.
- [ISO1926] ISO/IEC. *International Standard ISO/IEC 9126:2001*, 2001.
- [Jaz02] Mehdi Jazayeri. *On Architectural Stability and Evolution*. Ada-Europe 2002: 13-23, 2002.
- [JEE] Oracle. *Java Platform, Enterprise Edition (Java EE) Technical Documentation*.
<http://docs.oracle.com/javaee/>
- last visited 18.12.2011 -
- [Jen02] Scott Jenson. *The Simplicity Shift: Innovative Design Tactics in a Corporate World*. Cambridge University Press, 2002.
- [JLR00] Mehdi Jazayeri, Alexander Ran, Frank van der Linden. *Software Architecture for Product Families: Principles and Practice*. Addison Wesley, 2000.
- [Jos07] Nicolai M. Josuttis. *SOA in Practice. The Art of Distributed System Design*. O'Reilly. 2007.
- [KAB96] Rick Kazman, Gregory D. Abowd, Leonard J. Bass, Paul C. Clements: *Scenario-Based Analysis of Software Architecture*. IEEE Software 13(6): 47-55 1996.
- [Kan09] Udo Kannengiesser. *Process Flexibility: A Design View and Specification Schema*. EMISA 2009: 111-124, 2009.
- [Kan10] Udo Kannengiesser. *Towards a Methodology for Flexible Process Specification*. Enterprise Modelling and Information Systems Architectures 5(3): 44-63, 2010.
- [KBS04] Dirk Krafzig, Karl Banke, Dirk Slama. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice Hall, 2004.

- [KKN11] Thorsten Keuler, Jens Knodel, Matthias Naab. *Whitepaper – Architecture-Centric Software and Systems Engineering*. Fraunhofer ACES. IESE-Report No. 079.11/E, 2011.
<http://publica.fraunhofer.de/eprints/urn:nbn:de:0011-n-1863619.pdf>
- last visited 16.12.2011 -
- [KL10] Ronny Kolb, Frank van der Linden: *Point/Counterpoint*. IEEE Software 27(3): 56-59, 2010.
- [KMN06] Jens Knodel, Dirk Muthig, Matthias Naab. *Understanding Software Architectures by Visualization - An Experiment with Graphical Elements*. 13th Working Conference on Reverse Engineering (WCRE), 2006.
- [KMN+06] Jens Knodel, Dirk Muthig, Matthias Naab, Mikael Lindvall. *Static Evaluation of Software Architectures*. 10th European Conference on Software Maintenance and Reengineering (CSMR), 2006.
- [KMN08] Jens Knodel, Dirk Muthig, Matthias Naab. *An experiment on the role of graphical elements in architecture visualization*. Journal of Empirical Software Engineering, 13(6): 693-726, 2008.
- [Kno11] Jens Knodel. *Sustainable Structures in Software Implementations by Live Compliance Checking*. Fraunhofer Verlag, 2011.
- [Kru95] Philippe Kruchten. *Architectural Blueprints – The “4+1” View Model of Software Architecture*. IEEE Software 12(6), November 1995, pp. 42-50.
- [Kru03] Philippe Kruchten. *The Rational Unified Process: An Introduction*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
- [Kru10] Philippe Kruchten. *Software architecture and agile software development: a clash of two cultures?* ICSE (2) 2010: 497-498, 2010.
- [Lag07] Robert Lagerström. *Analyzing System Maintainability using Enterprise Architecture Models*. In Journal of Enterprise Architecture, vol. 3, no. 4, pp. 33-41, Nov. 2007.
- [LBV+02] Nico H. Lassing, PerOlof Bengtsson, Hans van Vliet, Jan Bosch. *Experiences with ALMA: Architecture-Level Modifiability Analysis*. Journal of Systems and Software 61(1): 47-57, 2002.
- [Len11] Stefan Lenz. *Business Architektur - Transparenz für das Business/IT-Alignment*.
http://www.stefan-lenz.ch/bit_glossar/89.html
- last visited 03.08.2011 -
- [LFJ+09] Robert Lagerström, Ulrik Franke, Pontus Johnson, and Johan Ullberg. *A Method for Creating Enterprise Architecture Metamodels – Applied to Systems Modifiability Analysis*. In International Journal of Computer Science & Applications, vol. 6, no. 5, pp. 89-120, Dec. 2009.
- [Lin00] David S. Linthicum. *Enterprise Application Integration*. Addison-Wesley Longman Ltd. Essex, UK, 2000.
- [LJE10] Robert Lagerström, Pontus Johnson, and Mathias Ekstedt. *Architecture Analysis of Enterprise Systems Modifiability – A Metamodel for Software Change Cost Estimation*. Software Quality Journal, vol.18, pp. 437-468, 2010.
- [LJH10] Robert Lagerström, Pontus Johnson, and David Höök. *Architecture Analysis of Enterprise Systems Modifiability – Models, Analysis, and Validation*. Journal of Systems and Software, vol. 83, no. 8, pp. 1387-1403, 2010.

- [LMN10] Jaejoon Lee, Dirk Muthig, Matthias Naab. *A feature-oriented approach for developing reusable product line assets of service-based systems*. Journal of Systems and Software 83(7). pp. 1123-1136, 2010.
- [LMN08] Jaejoon Lee, Dirk Muthig, Matthias Naab. *An Approach for Developing Service Oriented Product Lines*. 12th International Conference on Software Product Lines (SPLC), 2008.
- [LMS+07] Grace A. Lewis, Edwin Morris, Soumya Simanta, Lutz Wrage. *Common Misconceptions about Service-Oriented Architecture*. Proceedings of the Sixth International IEEE Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007.
- [LRV99a] Nico H. Lassing, Daan B. B. Rijsenbrij, Hans van Vliet: *Towards a Broader View on Software Architecture Analysis of Flexibility*. APSEC 1999: 238-245, 1999.
- [LRV99b] Nico Lassing, Daan Rijsenbrij, Hans van Vliet. *The Goal of Software Architecture Analysis: Confidence Building or Risk Assessment*. In: Proceedings of the First BeNeLux conference on Software Architecture. 1999.
- [LRV99c] Nico Lassing, Daan Rijsenbrij, Hans van Vliet. *Flexibility of the ComBAD Architecture*. 1st International Working Conference on Software Architecture (WICSA), 1999.
- [LRV01] Nico H. Lassing, Daan B. B. Rijsenbrij, Johannes C. van Vliet. *Viewpoints on Modifiability*. International Journal of Software Engineering and Knowledge Engineering 11(4): 453-478, 2001.
- [LRV03] Nico H. Lassing, Daan B. B. Rijsenbrij, Hans van Vliet. *How well can we predict changes at architecture design time?* Journal of Systems and Software 65(2): 141-153, 2003.
- [LSK07] Grace A. Lewis, Dennis B. Smith, Kostas Kontogiannis, Scott R. Tilley, Mira Kajko-Mattsson, Ned Chapin: *A Research Agenda for Maintenance & Evolution of SOA-Based Systems*. ICSM 2007: 481-484, 2007.
- [LSR07] Frank J. van der Linden, Klaus Schmid, Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer, 2007.
- [LSY11a] Lufthansa Systems. *Passenger Airline Solutions*. 2011.
<http://www.lhsystems.com/solutions/airline-solutions/passenger-airline-solutions/index.htm>
- last visited 18.12.2011 -
- [LSY11b] Lufthansa Systems. *GroundSolutions/Kiosk*. 2011
<http://www.lhsystems.com/solutions/airline-solutions/passenger-airline-solutions/ground-solutions-kiosk.htm>
- last visited 18.12.2011 -
- [Lub07] Boris Lublinsky. *Defining SOA as an architectural style*. IBM DeveloperWorks.
<http://www.ibm.com/developerworks/architecture/library/ar-soastyle>
- last visited 20.12.2011 -
- [Man09] Anne Thomas Manes. *SOA is Dead; Long Live Services*. 2009.
<http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html>
- last visited 31.10.2011 -
- [Mas07] Dieter Masak. *SOA?: Serviceorientierung in Business und Software*. Springer, 2007.

- [MER10] Wolfgang Martin, Julian Eckert, Nicolas Repp. *SOA Check 2010*. <http://www.soa-check.eu>
- last visited 04.08.2011 –
- [MG09] Peter Mell, Tim Grance. *The NIST Definition of Cloud Computing*. National Institute of Standards and Technology. Volume: 53, Issue: 6. 2009.
- [MM98] Tom Mens, Kim Mens: *Assessing the Evolvability of Software Architectures*. ECOOP Workshops 1998: 54-55, 1998.
- [Moo08] Leon Moonen. *Dealing with Crosscutting Concerns in Existing Software*. Frontiers of Software Maintenance, 2008. FoSM 2008.
- [Naa08] Matthias Naab. *SOA in Practice – The Most Popular Misconceptions*. Software Technology Initiative: Annual Event, Kaiserslautern, 2008.
- [Naa09] Matthias Naab. *Achieving True Flexibility of SOA-Based Information Systems by Adopting Practices from Product Line Engineering*. Doctoral Symposium of 13th International Conference on Software Product Lines (SPLC), 2009.
- [Naa11] Matthias Naab. *Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems*. 5th European Conference on Software Architecture (ECSA). Essen, Germany, 2011.
- [NAB11] Elisa Yumi Nakagawa, Pablo Oliveira Antonino, Martin Becker. *Reference Architecture and Product Line Architecture: A Subtle But Critical Difference*. ECSA 2011: 207-211, 2011.
- [NHJ05] Roshanak Nilchiani, Daniel Hastings, Carole Joppin. *Calculations of Flexibility in Space Systems*. INCOSE 15th International Symposium Rochester NY (2005) Issue: 617, Publisher: MIT, Pages: 1-19, 2005.
- [Nil05] Roshanak Nilchiani. *Measuring the Value of Space Systems Flexibility: A Comprehensive Six-element Framework*. PhD Thesis, Massachusetts Institute of Technology, 2005.
- [NM10] Matthias Naab, Dirk Muthig. *Designing Flexible Architectures for Product Lines Dominated by Open Variability*. IESE-Report No. 062.10/E, 2010.
- [NP07] Yefim V. Natis, Massimo Pezzini. *Twelve Common SOA Mistakes and How to Avoid Them*. Gartner Research. ID Number: G00152446, 2007.
- [OAS06] OASIS. *Reference Model for Service Oriented Architecture 1.0*. 2006, <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>
- last visited on 31.10.2011 –
- [OAS09] OASIS. *Reference Architecture Foundation for Service Oriented Architecture 1.0*. 2009, <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-cd-02.pdf>
- last visited on 31.10.2011 –
- [OKK07a] Ipek Ozkaya, Rick Kazman, Mark Klein. *Quality-Attribute Based Economic Valuation of Architectural Patterns*. 29th International Conference on Software Engineering Workshops(ICSEW'07), 2007.
- [OKK07b] Ipek Ozkaya, Rick Kazman, Mark Klein. *Quality-Attribute Based Economic Valuation of Architectural Patterns*. SEI Technical Report: CMU/SEI-2007-TR-003, 2007
- [OMG09] Object Management Group (OMG). *Service oriented architecture Modeling Language (SoaML) Beta1, Specification for the UML Profile and Metamodel for Services (UPMS)*, 2009.
<http://www.omg.org/spec/SoaML/1.0/Beta1>

- [OS03] Andreas Oberweis, Wolffried Stucky. *Flexibilität in betrieblichen Informationssystemen*. Informationswirtschaft: Ein Sektor mit Zukunft: 333-345, 2003.
- [Palladio] Palladio. *The Quality Software People. Software Architecture Simulator*. <http://www.palladio-simulator.com>
- last visited 02.01.2012 -
- [Par72] David Parnas. *On the Criteria To Be Used in Decomposing Systems into Modules*. Commun. ACM 15(12): 1053-1058, 1972.
- [Par79] David Parnas. *Designing Software for Ease of Extension and Contraction*. IEEE Trans. Software Eng. 5(2): 128-138, 1979.
- [Par94] David Parnas. *Software Aging*. International Conference on Software Engineering (ICSE) 1994: 279-287, 1994.
- [PK04] Barbara Paech, Kirstin Kohler. *Task-driven Requirements in Object-oriented Development*. Perspectives on Software Engineering. Kluwer Academic Publishers, 2004.
- [PLN+07] Pontus Johnson, Robert Lagerström, Per Närman, Mårten Simonsson, *Enterprise Architecture Analysis with Extended Influence Diagrams*. In Information Systems Frontiers, vol. 9, no. 2, pp. 163-180, May 2007.
- [Pro11] Progress Software. *SOA Misconceptions*. 2011.
<http://web.progress.com/en/soa-misconceptions.html>
- last visited on 31.10.2011 -
- [PW92] Dewayne E. Perry, Alexander L. Wolf. *Foundations for the study of software architecture* (Vol. 17, pp. 40-52): ACM, 1992.
- [RBB+11] Ralf Reussner, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Kozirolek, Heiko Kozirolek, Klaus Krogmann, Michael Kuperberg. *The Palladio Component Model*. Karlsruhe Reports in Informatics 2011,14, 2011.
- [RG08] Banani Roy, T.C. Nicholas Graham. *Methods for Evaluating Software Architecture: A Survey*. Technical Report No. 2008-545, School of Computing, Queen's University at Kingston, Ontario, Canada, 2008.
- [RH06] Ralf Reussner, Wilhelm Hasselbring. *Handbuch der Software-Architektur*. Dpunkt Verlag, 2006.
- [RRH08] Adam M. Ross, A., Donna H. Rhodes, and Daniel E. Hastings. *Defining Changeability: Reconciling Flexibility, Adaptability, Scalability, Modifiability, and Robustness for Maintaining Life Cycle Value*. Systems Engineering, Vol. 11, No. 3, pp.246-262, 2008.
- [RSS06] Gil Regev, Pnina Soffer, Rainer Schmidt. *Taxonomy of Flexibility in Business Processes*. BPMDS, 2006.
- [RW05] Nick Rozanski, Eoin Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison Wesley Longman, 2005.
- [RWC+03] Palani Rajan, Michael Van Wie, Matthew I. Campbell, Kristin L. Wood, Kevin N. Otto. *Design for Flexibility - Measures and Guidelines*. International Conference on Engineering Design, Stockholm, Sweden, 2003.
- [RWC+05] Palani Rajan, Michael Van Wie, Matthew I. Campbell, Kristin L. Wood, Kevin N. Otto. *An empirical foundation for product flexibility*. Elsevier, 2005.

- [Sal02] Joseph Homer Saleh. *Weaving time into system architecture : new perspectives on flexibility, spacecraft design lifetime, and on-orbit servicing*. PhD Thesis, Massachusetts Institute of Technology, 2002.
- [SB01] Ken Schwaber, Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.
- [Sch02] Klaus Schmid. *A comprehensive product line scoping approach and its validation*. International Conference on Software Engineering (ICSE) 2002: 593-603, 2002.
- [Sch04] Michael Schrage. *The Struggle to Define Agility*. www.cio.com, 2004.
- [Sel03] Bran Selic. *The pragmatics of model-driven development*. IEEE software, 20(5):19-25, 2003.
- [SG96] Mary Shaw, David Garlan. *Software architecture: perspectives on an emerging discipline*. Prentice-Hall, Inc., 1996.
- [SG11] Franz Schober, Judith Gebauer. *How much to spend on flexibility? Determining the value of information system flexibility*. Decision Support Systems 51(3): 638-647, 2011.
- [SHN01] Joseph H. Saleh, Daniel E. Hastings, Dava J. Newman. *Extracting The Essence Of Flexibility In System Design*. Proc. IEEE NASA/DoD Workshop on Evolvable Hardware, 2001.
- [SHN03] Joseph H Saleh, Daniel E Hastings, Dava J Newman. *Flexibility in system design and implications for aerospace systems*. Acta Astronautica, Volume 53, Issue 12, December 2003, Pages 927-944, 2003.
- [Sie04] Johannes Siedersleben. *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar*. dpunkt.verlag, 2004.
- [SMC74] W.P. Stevens, G.J. Myers, L.L. Constantine. *Structured Design*. IBM Systems Journal 13, 2, Pages 115-139, 1974.
- [Smi08] Kevin T. Smith. *Flexibility by Design – Adapting to changes at Runtime in SOA Implementations*, SOAInstitute, 2008.
<http://www.soainstitute.org/white-papers/white-paper/article/flexibility-by-design-adapting-to-changes-at-run-time-in-soa-implementations-1.html>
- last visited 17.12.2011 -
- [SMR+08] Helen Schonenberg, Ronny Mans, Nick Russell, Nataliya Mulyar, Wil M. P. van der Aalst. *Towards a Taxonomy of Process Flexibility*. CAiSE Forum 2008: 81-84, 2008.
- [SN96] Roy W. Schulte, Yefim V. Natis. *“Service Oriented” Architectures , Part 1*. Gartner Research Note SPA-401-068, 1996.
- [Sne95] Harry M. Sneed. *Estimating the costs of software maintenance tasks*. ICSM 1995: 168-181, 1995.
- [Som07] Ian Sommerville. *Software Engineering*. Pearson Studium, 2007.
- [Spr05] David Sprott. *Business Flexibility Through SOA*. CBDI Report, 2005.
<ftp://ftp.software.ibm.com/software/soa/pdf/CBDIWhitepaperBusinessFlexibilityThroughSOA.pdf>
- last visited 17.12.2011 -
- [SR09] Johannes Stammel, Ralf Reussner. *KAMP: Karlsruhe Architectural Maintainability Prediction*. Proceedings des 1. Workshop des GI-Arbeitskreises Langlebige Softwaresysteme (L2S2): "Design for Future - Langlebige Softwaresysteme", Karlsruhe, Germany, 2009.
- [Suh05] Eun Suk Suh. *Flexible Product Platforms*. PhD Thesis, Massachusetts Institute of Technology, 2005.

- [SWV+08] Nirav B. Shah, Jennifer Wilds, Lauren Viscito, Adam M. Ross, Daniel E. Hastings. *Quantifying Flexibility for Architecting Changeable Systems*, Conference on Systems Engineering Research 2008, Los Angeles, CA, 2008.
- [Til08] Stefan Tilkov. *Von 0 auf SOA in 10 Schritten*, W-JAX 2008, <http://www.innoq.com/blog/st/presentations/2008/2008-11-03-SOA10--WJAX.pdf>
- last visited 31.10.2011-
- [TMD09] Richard. N. Taylor, Nenad. Medvidovic, and Eric. M. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, January 2009.
- [TSW09] Andrew H. Tilstra, Carolyn C. Seepersad, Kristin L. Wood. *Analysis of Product Flexibility for Future Evolution Based on Design Guidelines and a High-Definition Design Structure Matrix*. Proceedings of the ASME 2009 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, IDETC/CIE 2009, August 30 - September 2, 2009, San Diego, California, USA, 2009.
- [VDG08] Karina Villela, Jörg Dörr, Anne Gross. *Proactively Managing the Evolution of Embedded System Requirements*. RE 2008: 13-22, 2008.
- [VEG08] Karina Barreto Villela, Michael Eisenbarth, Anne Gross. *Supporting Software Evolution in the Context of Product Lines*. IESE-Report No. 015.08/E, 2008.
- [W3C] World Wide Web Consortium (W3C). <http://www.w3.org>
- last visited 19.12.2011 -
- [WDF08] Stefan Wagner, Florian Deißenböck, Martin Feilkas, Elmar Jürgens. *Software-Qualitätsmodelle in der Praxis: Erfahrungen mit aktivitätsbasierten Modellen*. Workshop Software-Qualitätsmodellierung und -bewertung (SQMB '08), 2008.
- [Wes07] Matthias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
- [Wit07] Christoph Witte. *IT denkt Business*. <http://www.computerwoche.de/cio-des-jahres/2007/1848346/>
- last visited 03.08.2011 -
- [WRH+00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslen. *Experimentation in software engineering: an Introduction*. Kluwer Academic Publishers, 2000.
- [Zim09] Olaf Zimmermann. *An Architectural Decision Modeling Framework for Service-Oriented Architecture Design*. Dissertation, dissertation.de, 2009.
- [ZYX+02] Jianjun Zhao, Hongji Yang, Liming Xiang, Baowen Xu. *Change impact analysis to support architectural evolution*. Journal of Software Maintenance and Evolution: Research and Practice - Special Issue: Separation of Concerns for Software Evolution. Volume 14, Issue 5, pages 317–333, September/October 2002, 2002.

Appendix A List of Abbreviations

ACES	Architecture-Centric Engineering Solutions
ALMA	Architecture-Level Modifiability Analysis
BAM	Business Activity Monitoring
BPEL	Business Process Execution Language
BLA	Business-Logic-Agnostic
BLM	Business Logic Mapping
BLS	Business-Logic-Specific
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Modeling Notation
BRM	Business Rule Management
CORBA	Common Object Request Broker Architecture
EA	Enterprise Architect
EAM	Enterprise Architecture Management
EDA	Event-Driven Architecture
ESB	Enterprise Service Bus
GQM	Goal-Question-Metric
IT	Information Technology
IESE	Institute for Experimental Software Engineering
LOC	Lines of Code
OASIS	Organization for the Advancement of Structured Information Standards
OMG	Object Management Group

OSGi	Open Services Gateway initiative
PLE	Product Line Engineering
REST	Representational State Transfer
SCA	Service Component Architecture
SEI	Software Engineering Institute
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOE	Service-Oriented Engineering
UDDI	Universal Description, Discovery and Integration
UI	User Interface
UML	Unified Modeling Language
WCF	Windows Communication Foundation
WSDL	Web Service Description Language
XPDL	XML Process Definition Language
XML	eXtensible Markup Language

Appendix B Experiment Material

In the following, we provide the experiment material that was given to the experiment participants in the experiment described in Section 7.

Please note that the material provided here contains the entirety of material sheets. Depending on the groups A and B, the respective material was handed out according to the experiment design described in Section 7.2.2.2.

The differences between material for groups A and B are the following:

- Different Task Descriptions (pages 4(A) and 4(B))
- Different Debriefing Questionnaires (pages 5(A) and 5(B))
- Material 4 handed out at different points in time (group A got it after conducting the described tasks, group B received it directly, according to the experimental design)



ID: _____

Practical Course: Architecture Experiment - Procedures

First of all, thank you for participating in the architecture experiment!

Please note that the evaluation of this experiment will be done anonymously. However, please make sure that you put your ID on all artifacts you produce.

Please read this document carefully and contact your experimenter in case you have fundamental problems in understanding the given information or tasks. Typically, the preparation takes about 10 minutes.

Please don't talk about the experiment to other participants of the course before all conducted the experiment!

Please try to perform the task as fast as possible but aim at achieving a high quality. The maximal allotted time for executing the experiment is 90 min.

Preparation

- Note preparation start time here (e.g. 10:00): _____
- Read the Introduction (page 2)
- Fill in the Briefing Questionnaire (page 3)
- Note preparation stop time here (e.g. 10:15): _____

Execution

- Read the task description and conduct the tasks (page 4)

Finalization

- Note finalization start time here (e.g. 11:30) _____
- Fill in the Debriefing Questionnaire (page 5)

Introduction to the Architecture Experiment

Purpose:

This experiment investigates how the flexibility of systems is impacted by architecture design.

Background on flexibility and architecture:

Therefore, we start with a definition of flexibility as its understanding is crucial for the experiment.

Definition of flexibility: Intuitively, flexibility is the degree to which a system supports a set of anticipated changes to its requirements. Quality requirements with a similar meaning are modifiability, maintainability, or extensibility.

That means, a system is flexible with respect to a certain anticipated requirement, if it can be changed to fulfill this requirement with relatively low effort and cost. Making the right architecture decisions determines the flexibility of a system. It's an architect's responsibility to design architectures with an appropriate degree of flexibility. But how is this done?

Designing architectures for flexibility: The key idea behind designing for flexibility is to find a solution, which strongly limits the impact of the anticipated changes (principle: localization of change impact).

Context in the practical course:

The experiment takes place in the context of the 2011 Software Engineering practical course at IESE. It is embedded in the development activities of the John Deere project. The experiment is based on the results of iteration 1 in the development. Therefore, a part of the architecture as designed by the student team in iteration 1 was selected and modeled in an appropriate notation as input for the experiment.

It is not a problem for the experiment or the further activities in the project if the architecture as you find it modeled in the material here does not fully comply to your ideas of the project.

Your role in the experiment:

In the experiment, you act as an architect (independent of the role you have in the project). Your task will be to change / extend an existing architecture from the project. The task is deliberately kept easy so that everyone should be able to conduct them.



ID: _____

Briefing Questionnaire

Remark

The following questions help us evaluate the results of the practical exercise in more detail. The information you provide is important to evaluate the applicability and usefulness of the architecture design. Your answers will be treated anonymously. The ID you provide will only be used to connect your questionnaire to your results.

Please answer the questions as complete and honest as possible. Thank you for your support!

Background Information

How old are you? _____

What is your major subject of study (e.g. computer science)? _____

In which semester of your studies are you at the moment? _____

Did you attend lectures on architecture at university (Yes / No)? _____

Which role do you have in the course (e.g. architect, tester)? _____

Please indicate in the following categories *your previous experiences*:

	No experience				High experience
	1	2	3	4	5
Working in software development projects					
Reading and using UML					
Architecting: Defining software architectures					

Task Description

Summarizing, your task is to take the role of an architect and change / extend the existing architecture from the project in a way, that the system becomes flexible with respect to certain anticipated flexibility requirements. Please note that you do NOT have to design the system for directly fulfilling the stated requirements, but you have to design the system for being easily changeable to fulfill the stated requirements.

You have the following *material as input* for your work:

- *Architecture Documentation* of the system to be changed / extended (Material 1)
- *Architecture Flexibility Requirements*: the flexibility requirements to be incorporated in the architecture (Material 2)
- *Architecture Modeling Notation & Example*: Support for your modeling by explaining the notation used and giving examples (Material 3)

Please *conduct* the following *activities*:

- Carefully read and understand the input material
- Change / extend the input architecture in a way that it fulfills the flexibility requirements well (little effort and cost needed to conduct the described changes)
- Document your resulting architecture on the prepared sheets ("Your Task Results").
 - Please describe the full resulting architecture, not only changes
 - Please follow the given architecture modeling notation and the example
 - Please provide for every scenario an independent solution description (that is, don't define 1 architecture fulfilling all scenarios, but 3 architectures each fulfilling the respective scenario)
 - Please add a brief description of your ideas and architecture decisions

After finishing the activities, please *ask your experimenter for one more material* (Material 4: Architecture Change Impact Notation & Example).

Please *conduct the activity*, which is *described on Material 4*.

Task Description

Summarizing, your task is to take the role of an architect and change / extend the existing architecture from the project in a way, that the system becomes flexible with respect to certain anticipated flexibility requirements. Please note that you do NOT have to design the system for directly fulfilling the stated requirements, but you have to design the system for being easily changeable to fulfill the stated requirements.

You have the following *material as input* for your work:


- *Architecture Documentation* of the system to be changed / extended (Material 1)
- *Architecture Flexibility Requirements*: the flexibility requirements to be incorporated in the architecture (Material 2)
- *Architecture Modeling Notation & Example*: Support for your modeling by explaining the notation used and giving examples (Material 3)
- *Architecture Change Impact Notation & Example*: Support about an additional architectural view, which you are asked to create. Explained with notation and example (Material 4)


Please *conduct* the following *activities*:

- Carefully read and understand the input material
- Change / extend the input architecture in a way that it fulfills the flexibility requirements well (little effort and cost needed to conduct the described changes)
Please keep in mind both, Material 3 and Material 4!
- Document your resulting architecture on the prepared sheets ("Your Task Results").
 - Please document the resulting *architectural structure* (as described in Material 3)!
 - Please document the *expected change impact* (as described in Material 4)!
 - Please describe the full resulting architecture, not only changes!
 - Please follow the given architecture modeling notations and the examples!
 - Please provide for every scenario an independent solution description (that is, don't define 1 architecture fulfilling all scenarios, but 3 architectures each fulfilling the respective scenario)
 - Please add a brief description of your ideas and architecture decisions


Debriefing Questionnaire

Please answer the following concluding questions about the experiment conduction.

	Not Good  Good				
	1	2	3	4	5
How well did you <i>understand</i> the tasks?					

	Easy  Difficult				
	1	2	3	4	5
How <i>difficult</i> did you perceive the tasks?					


	Low Quality  High Quality				
	1	2	3	4	5
How do you estimate the <i>quality of your results</i> (high flexibility)?					


	Fully Disagree  Fully Agree				
	1	2	3	4	5
During architecture design, I changed my solutions when I recognized that the flexibility is insufficient					
After modeling the change impact, I would have liked to change my architecture solutions as I recognized better possibilities					
Modeling the change impact during architecture design would have helped me to come up with a better solution					

Do you have any further *comments* about the experiment?


Debriefing Questionnaire

Please answer the following concluding questions about the experiment conduction.

	Not Good					Good
	1	2	3	4	5	
How well did you <i>understand</i> the tasks?						

	Easy					Difficult
	1	2	3	4	5	
How <i>difficult</i> did you perceive the tasks?						

	Low Quality					High Quality
	1	2	3	4	5	
How do you estimate the <i>quality of your results</i> (high flexibility)?						

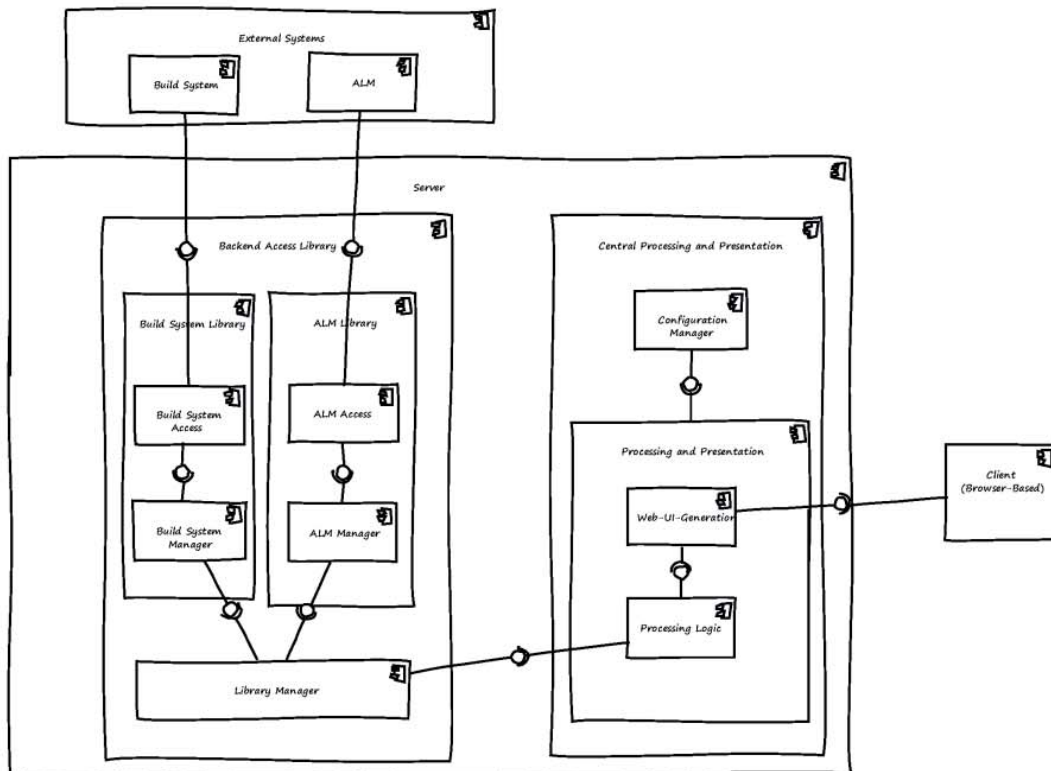
	Fully Disagree					Fully Agree
	1	2	3	4	5	
During architecture design, I changed my solutions when I recognized that the flexibility is insufficient						
After reasoning about / modeling the change impact, I changed my architecture solutions as I recognized better possibilities						
Modeling the change impact during architecture design helped me to come up with a better solution						

Do you have any further *comments* about the experiment?

Material 1: Architecture Documentation Input

Please note that this architectural input is based on the current architecture document of the practical course but might have also slight differences, which are only valid for the experiment! Only one view showing the main structure of the software is depicted. The design decisions and further information is to be assumed as described in the architecture document.

On Material 3 you will find more information on the notation of architecture documentation.



Material 1

Material 2: Architecture Flexibility Requirements

Please note that the description of the scenarios is not tabular (as you might know it) but contains the same information.

Please note that these scenarios are only for the experiment, they are not known to be requirements of the customer!

Please provide for every scenario an independent solution description (that is, don't define 1 architecture fulfilling all scenarios, but 3 architectures each fulfilling the respective scenario)

Scenario 1: Support for mobile end devices

Besides the user interface based on browser technology the system should be also extended with displaying support on mobile devices. In order to maintain a high usability and native look and feel, the mobile devices should be equipped with native apps for the respective platforms. For a particular mobile technology (e.g. iOS, Android, ...) an integration of the app with the system should be possible with no or only minimal changes to the system.

Scenario 2: Usage of new data field

Jenkins (a build system) as one of the key tools delivering data is expected to constantly evolve and come up with new data fields, which might be also interesting to be processed and presented in our system. It should be possible to integrate a new data field of Jenkins in our system, which means that it has to be read from Jenkins, can be used in the analyses and processings, and might be represented directly in the user interface. Changing our system with respect to the extension of a new data field should be possible with minimum change effort.

Scenario 3: Extensible evaluation and representation

Besides the simple presentation logic of data from external systems, also more sophisticated analyses will be required in our system. Such analyses should be available in a widget-style (similar to Windows 7 widgets), which can be integrated in the user interface and work on all available input data. Our system should be flexible in a way that such widgets can be realized as self-contained elements. That means, one can develop a new widget consisting of analyses logic and a graphical representation, which can be added to our system with no or only minimum change effort.

Material 3: Architecture Modeling Notation & Example

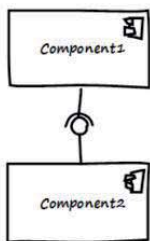
Please note that this is a simplified notation for architecture modeling in the experiment. Only 1 architectural viewtype is used (e.g. no deployment view), showing the main structure of the system.

When you model your architectural results, please add a brief description of your ideas and architecture decisions.



Computational Component

(Assumes to represent the runtime entity as well as development entity)



Connector

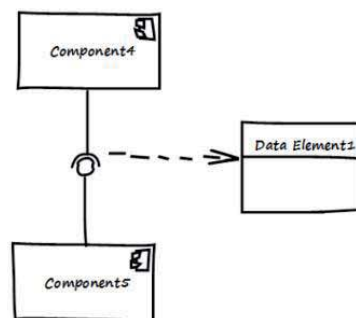
(Relationship between Component1 and Component2, mainly in the form of a „uses“-relationship)



You might also model data elements, which were not yet part of the input architecture

Data Element (Usage)

(Represents a data element, which is used by Component3, e.g. for processing, transforming, or representing the data)



Data Element (Transport)

(Represents a data element, which is transported between Component4 and Component5)

Material 4: Architecture Change Impact Notation & Example

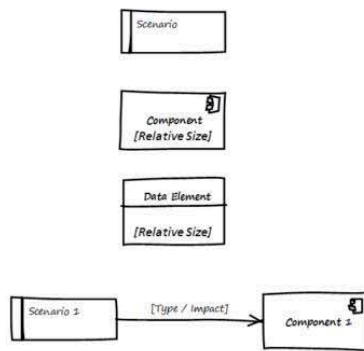
The architecture change impact notation is a new and additional architectural viewtype, which intends to explicitly model the impact of anticipated changes during architecture design. It is an additional task for an architect to model this view.

Activity to be conducted with Material 4:

Please model the change impact of the described flexibility requirements as you expect them for the architecture you designed!

Please provide for each flexibility scenario a single change impact diagram!

Please draw your results on the provided sheets ("Your Task Results").



Scenario

(Represents a Flexibility Scenario, for which the change impact is modeled)

Computational Component

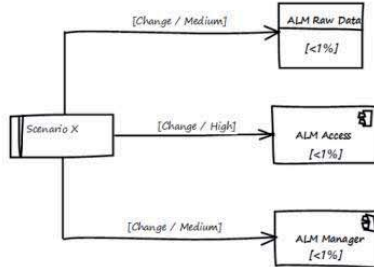
(Estimated relative size of component code in overall system)

Data Element

(Estimated relative size of data element code in overall system)

Change Impact

(If the flexibility described in Scenario 1 is exploited, a change of Component 1 is necessary)
 Type of change on component: (add / change / delete)
 Impact of change on component: (low / medium / high)



Example

Scenario X be: A new ALM system becomes available and has to be integrated into the system (replacing the old one). Only minimum effort should be necessary to integrate the new ALM system.

As the new ALM system might have other data formats, the data structures might need to be changed (with medium impact on the code). Further changes are necessary to the ALM Access component (high impact) and the ALM Manager component (medium impact).

Each impacted element is estimated to have a relative size of <1% in the overall system code base).

Your Task Results

Results

Appendix C Experiment Raw Data

In the following, we provide the raw data of the experiment. This is mainly the answers to questionnaires and evaluation of flexibility measurement results.

The data analysis procedure is described in detail in Section 7.2.3.1. The results are grouped by the experiment groups A and B; each participant is identified by a number in their group.

On the following pages, we always contrast groups A and B on one page. The results are organized in the following groups, each on one page:

- Briefing questionnaire and time needed
- Debriefing questionnaire
- Flexibility measurement results

	<i>Preparation Time</i>	<i>Execution Time</i>	<i>Age</i>	<i>Major subject</i>	<i>Semester (in Master studies)</i>	<i>Architecture lectures?</i>	<i>Role in course</i>	<i>Working in dev projects</i>	<i>Reading and using UML</i>	<i>Architecting</i>
A1	10	70	24	CS	2	Y	Tester	3	3	2
A2	7	82	25	SE	2	Y	UI	4	1	2
A3	20	75	23	CS	3	Y	UI	2	4	1
A4	13	65	27	CS	3	Y	RE	3	3	2
A5	5	78	26	TC	3	Y	Project Manager	1	1	1
A6	5	60	30	SE	3	Y	Tester	4	3	3
A7	12	78	23	CS	2	Y	RE	3	3	2
A8	10	62	24	SE	3	Y	Architect	4	3	3
Avg.	10,25	71,25	25,25		2,63			3,00	2,63	2,00

Table 18: Experiment raw data: Group A – Briefing Questionnaire

	<i>Preparation Time</i>	<i>Execution Time</i>	<i>Age</i>	<i>Major subject</i>	<i>Semester (in Master studies)</i>	<i>Architecture lectures?</i>	<i>Role in course</i>	<i>Working in dev projects</i>	<i>Reading and using UML</i>	<i>Architecting</i>
B1	20	77	24	SE	3	N	UI	2	1	1
B2	3	29	25	SE	3	Y	RE	5	5	1
B3	14	78	26	CS	5	N	RE	3	3	2
B4	4	83	24	CS	3	Y	Architect	3	4	3
B5	4	69	25	CS	4	Y	Architect	1	2	1
B6	6	63	23	CS	3	Y	Developer	4	3	2
B7	5	71	27	CS	3	N	Project Manager	4	1	1
B8	20	70	27	CS	3	Y	Architect	4	4	4
B9	4	41	27	SE	3	Y	Developer	4	4	4
Avg.	8,89	64,56	25,33		3,33			3,33	3,00	2,11

Table 19: Experiment raw data: Group B – Briefing Questionnaire

Experiment Raw Data

	<i>How well did you understand the tasks?</i>	<i>How difficult did you perceive the tasks?</i>	<i>How do you estimate the quality of your results?</i>	<i>Changed my solutions when recognized insufficient flexibility</i>	<i>After modeling change impact I would have liked to change architecture</i>	<i>Modeling change impact (would have) helped me</i>
A1	5	4	3	4	4	5
A2	4	3	3	4	3	4
A3	3	3	2	3	3	4
A4	1	4	2	4	4	4
A5	4	2	4	1	1	3
A6	4	2	3	5	1	4
A7	4	1	4	2	3	3
A8	3	3	3	4	3	5
Avg.	3,50	2,75	3,00	3,38	2,75	4,00

Table 20: Experiment raw data: Group A – Debriefing Questionnaire

	<i>How well did you understand the tasks?</i>	<i>How difficult did you perceive the tasks?</i>	<i>How do you estimate the quality of your results?</i>	<i>Changed my solutions when recognized insufficient flexibility</i>	<i>After modeling change impact I would have liked to change architecture</i>	<i>Modeling change impact (would have) helped me</i>
B1	3	4	3	4	4	4
B2	5	1	4	1	1	1
B3	3	3	3	2	4	3
B4	4	3	4	3	1	4
B5	3	4	3	4	1	4
B6	4	2	4	1	1	1
B7	3	3	3	2	2	3
B8	4	3	3	4	3	4
B9	3	3	3	2	2	4
Avg.	3,56	2,89	3,33	2,56	2,11	3,11

Table 21: Experiment raw data: Group B – Debriefing Questionnaire

	1: Adequacy independent of flexibility (Components)	1: Adequacy independent of flexibility (Change impact)	1: Flexibility	2: Adequacy independent of flexibility (Components)	2: Adequacy independent of flexibility (Change impact)	2: Flexibility	3: Adequacy independent of flexibility (Components)	3: Adequacy independent of flexibility (Change impact)	3: Flexibility
A1	2	5		5	3	3	1	3	
A2	2	5		1	1		2	3	
A3	1	2		1	1		3	3	4
A4	3	3	3	3	3	3	3	3	3
A5	1	5		1	5		3	5	2
A6	3	3	4	3	3	3	3	2	
A7	1	3		3	3	3	1	3	
A8	1	5		3	5	3	3	4	3
Avg.	1,75	3,88	3,50	2,50	3,00	3,00	2,38	3,25	3,00

Table 22: Experiment raw data: Group A – Flexibility Results

	1: Adequacy independent of flexibility (Components)	1: Adequacy independent of flexibility (Change impact)	1: Flexibility	2: Adequacy independent of flexibility (Components)	2: Adequacy independent of flexibility (Change impact)	2: Flexibility	3: Adequacy independent of flexibility (Components)	3: Adequacy independent of flexibility (Change impact)	3: Flexibility
B1	2	3		1	2		2	2	
B2	3	3	4	5	5	4	3	5	4
B3	2	4		2	3		2	4	
B4	5	5	5	4	5	4	5	5	5
B5	2	4		4	5	3	3	3	4
B6	5	3	5	5	2		1	2	
B7	2	3		1	4		1	3	
B8	2	3		2	4		2	4	
B9	5	5	5	5	5	3	2	3	
Avg.	3,11	3,67	4,75	3,22	3,89	3,50	2,33	3,44	4,33

Table 23: Experiment raw data: Group B – Flexibility Results

Lebenslauf

Persönliche Daten

Name	Matthias Naab
Anschrift	Zollamtstraße 64 67663 Kaiserslautern
Geburtsdatum und -ort	25.07.1979 in Dahn
Familienstand	verheiratet, 1 Kind

Werdegang

1986 - 1990	Grundschule Dahn
1990 - 1999	Otfried-von-Weißenburg-Gymnasium, Dahn (Abitur)
1999 - 2000	Zivildienst, Jugendherberge Dahn
2000 - 2005	Studium der Informatik, Universität Kaiserslautern (Diplom)
seit 2005	Wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Experimentelles Software Engineering (IESE), Kaiserslautern

Kaiserslautern, den 02.10.2012

PhD Theses in Experimental Software Engineering

- Volume 1** **Oliver Laitenberger** (2000), *Cost-Effective Detection of Software Defects Through Perspective-based Inspections*
- Volume 2** **Christian Bunse** (2000), *Pattern-Based Refinement and Translation of Object-Oriented Models to Code*
- Volume 3** **Andreas Birk** (2000), *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*
- Volume 4** **Carsten Tautz** (2000), *Customizing Software Engineering Experience Management Systems to Organizational Needs*
- Volume 5** **Erik Kamsties** (2001), *Surfacing Ambiguity in Natural Language Requirements*
- Volume 6** **Christiane Differding** (2001), *Adaptive Measurement Plans for Software Development*
- Volume 7** **Isabella Wiczorek** (2001), *Improved Software Cost Estimation A Robust and Interpretable Modeling Method and a Comprehensive Empirical Investigation*
- Volume 8** **Dietmar Pfahl** (2001), *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*
- Volume 9** **Antje von Knethen** (2001), *Change-Oriented Requirements Traceability Support for Evolution of Embedded Systems*
- Volume 10** **Jürgen Münch** (2001), *Muster-basierte Erstellung von Software-Projektplänen*
- Volume 11** **Dirk Muthig** (2002), *A Light-weight Approach Facilitating an Evolutionary Transition Towards Software Product Lines*
- Volume 12** **Klaus Schmid** (2003), *Planning Software Reuse – A Disciplined Scoping Approach for Software Product Lines*
- Volume 13** **Jörg Zettel** (2003), *Anpassbare Methodenassistenz in CASE-Werkzeugen*
- Volume 14** **Ulrike Becker-Kornstaedt** (2004), *Prospect: a Method for Systematic Elicitation of Software Processes*
- Volume 15** **Joachim Bayer** (2004), *View-Based Software Documentation*
- Volume 16** **Markus Nick** (2005), *Experience Maintenance through Closed-Loop Feedback*

- Volume 17** **Jean-François Girard** (2005), *ADORE-AR: Software Architecture Reconstruction with Partitioning and Clustering*
- Volume 18** **Ramin Tavakoli Kolagari** (2006), *Requirements Engineering für Software-Produktlinien eingebetteter, technischer Systeme*
- Volume 19** **Dirk Hamann** (2006), *Towards an Integrated Approach for Software Process Improvement: Combining Software Process Assessment and Software Process Modeling*
- Volume 20** **Bernd Freimut** (2006), *MAGIC: A Hybrid Modeling Approach for Optimizing Inspection Cost-Effectiveness*
- Volume 21** **Mark Müller** (2006), *Analyzing Software Quality Assurance Strategies through Simulation. Development and Empirical Validation of a Simulation Model in an Industrial Software Product Line Organization*
- Volume 22** **Holger Diekmann** (2008), *Software Resource Consumption Engineering for Mass Produced Embedded System Families*
- Volume 23** **Adam Trendowicz** (2008), *Software Effort Estimation with Well-Founded Causal Models*
- Volume 24** **Jens Heidrich** (2008), *Goal-oriented Quantitative Software Project Control*
- Volume 25** **Alexis Ocampo** (2008), *The REMIS Approach to Rationale-based Support for Process Model Evolution*
- Volume 26** **Marcus Trapp** (2008), *Generating User Interfaces for Ambient Intelligence Systems; Introducing Client Types as Adaptation Factor*
- Volume 27** **Christian Denger** (2009), *SafeSpection – A Framework for Systematization and Customization of Software Hazard Identification by Applying Inspection Concepts*
- Volume 28** **Andreas Jedlitschka** (2009), *An Empirical Model of Software Managers' Information Needs for Software Engineering Technology Selection
A Framework to Support Experimentally-based Software Engineering Technology Selection*
- Volume 29** **Eric Ras** (2009), *Learning Spaces: Automatic Context-Aware Enrichment of Software Engineering Experience*
- Volume 30** **Isabel John** (2009), *Pattern-based Documentation Analysis for Software Product Lines*
- Volume 31** **Martín Soto** (2009), *The DeltaProcess Approach to Systematic Software Process Change Management*
- Volume 32** **Ove Armbrust** (2010), *The SCOPE Approach for Scoping Software Processes*

- Volume 33** **Thorsten Keuler** (2010), *An Aspect-Oriented Approach for Improving Architecture Design Efficiency*
- Volume 34** **Jörg Dörr** (2010), *Elicitation of a Complete Set of Non-Functional Requirements*
- Volume 35** **Jens Knodel** (2010), *Sustainable Structures in Software Implementations by Live Compliance Checking*
- Volume 36** **Thomas Patzke** (2011), *Sustainable Evolution of Product Line Infrastructure Code*
- Volume 37** **Ansgar Lamersdorf** (2011), *Model-based Decision Support of Task Allocation in Global Software Development*
- Volume 38** **Ralf Carbon** (2011), *Architecture-Centric Software Producibility Analysis*
- Volume 39** **Florian Schmidt** (2012), *Funktionale Absicherung kamerabasierter Aktiver Fahrerassistenzsysteme durch Hardware-in the-Loop-Tests*
- Volume 40** **Frank Elberzhager** (2012), *A Systematic Integration of Inspection and Testing Processes for Focusing Testing Activities*
- Volume 41** **Matthias Naab** (2012), *Enhancing Architecture Design Methods for Improved Flexibility in Long-Living Information Systems*

Software Engineering has become one of the major foci of Computer Science research in Kaiserslautern, Germany. Both the University of Kaiserslautern's Computer Science Department and the Fraunhofer Institute for Experimental Software Engineering (IESE) conduct research that subscribes to the development of complex software applications based on engineering principles. This requires system and process models for managing complexity, methods and techniques for ensuring product and process quality, and scalable formal methods for modeling and simulating system behavior. To understand the potential and limitations of these technologies, experiments need to be conducted for quantitative and qualitative evaluation and improvement. This line of software engineering research, which is based on the experimental scientific paradigm, is referred to as 'Experimental Software Engineering'.

In this series, we publish PhD theses from the Fraunhofer Institute for Experimental Software Engineering (IESE) and from the Software Engineering Research Groups of the Computer Science Department at the University of Kaiserslautern. PhD theses that originate elsewhere can be included, if accepted by the Editorial Board.

Editor-in-Chief: Prof. Dr. Dieter Rombach

Executive Director of Fraunhofer IESE and Head of the AGSE Group of the Computer Science Department, University of Kaiserslautern

Editorial Board Member: Prof. Dr. Peter Liggesmeyer

Scientific Director of Fraunhofer IESE and Head of the AGDE Group of the Computer Science Department, University of Kaiserslautern

Editorial Board Member: Prof. Dr. Frank Bomarius

Deputy Director of Fraunhofer IESE and Professor for Computer Science at the Department of Engineering, University of Applied Sciences, Kaiserslautern

ISBN 978-3-8396-0477-9



9 783839 604779